



Andreas Abraham, BSc

# **Estimating the Power Consumption of Virtual Machines and their Applications**

## **MASTER'S THESIS**

to achieve the university degree of

Diplom-Ingenieur

Master's degree programme: Software Development and Business Management

submitted to

**Graz University of Technology**

Supervisor

Assessor: Ass.Prof. Dipl.-Ing. Dr.techn. Christian Steger

Institute for Technical Informatics

Advisor: Ass.Prof. Dipl.-Ing. Dr.techn. Christian Steger  
Dr. Damian Dalton (University College Dublin)

Graz, August 2015

## **AFFIDAVIT**

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used. The text document uploaded to TUGRAZonline is identical to the present master's thesis dissertation.

---

Date

---

Signature

## Kurzfassung

Der steigende Trend von E-Commerce und Cloud Computing sind nur zwei Gründe des wachsenden Bedarfs an Datenzentren. Datenzentren gehören zu den größten globalen Energieverbrauchern im Bezug auf den Gesamtenergiekonsum weltweit. Mit dieser Entwicklung wächst jedoch auch der Energieverbrauch, der zum Betreiben neuer Datenzentren benötigt wird und der damit verbundene "Carbon Footprint" von Datenzentren gewinnt immer mehr an Bedeutung. Die steigende Anzahl an Datenzentren hat einen negativen Einfluss auf die Umwelt welcher auf den steigenden Energieverbrauch zurückzuführen ist.

Das University College Dublin (UCD) hat sich diese Problematik zum Thema gemacht und versucht Datenzentren grüner zu machen, sprich das Energiemanagement zu verbessern und dadurch weniger Energie zu verbrauchen. Der Fokus liegt bei dem Haupt-Energieverbraucher eines Datenzentrums, den Servern. Das Papillon Master-Client Projekt wurde entwickelt um den Energieverbrauch von Servern in einem Datenzentrum zu überwachen und zu Reporten. Der Energieverbrauch eines Servers wird am Master berechnet ohne zusätzliche Hardware einsetzen zu müssen. Ein immer beliebteres Tool in Datenzentren ist die Virtualisierung. Dies wird unter anderem dazu eingesetzt um die vorhandene Hardware effizienter zu nutzen. Virtuelle Maschinen wurden vom Papillon System nicht unterstützt.

Diese Arbeit beschäftigt sich mit dem ermitteln des Energieverbrauchs von virtuellen Maschinen ohne das Verwenden zusätzlicher Hardware, basierend auf dem bereits bestehenden Papillon Systems. Begonnen wurde mit einer ausführlichen Literatur Recherche gefolgt von einer Analyse des Papillon Systems. Nach experimentellen Test wurde das Design des zu implementierenden Systems geplant. Daraufhin folgte die Implementation sowie das Testen und Evaluieren des neu entwickelten Systems.

## **Abstract**

The rising trend of e-commerce and cloud computing are just two reasons for the growing demand for Data Centers (DCs). DCs are among the largest global energy consumers in relation to the total global energy consumption. The rising number of DCs pose an increasingly negative impact on the environment. This may be caused by the DCs themselves as well as by the power generation that is needed by the DCs. For this reason, the carbon footprint of DCs is increasingly gaining importance.

University College Dublin has made the main energy consumers of data centers, which are servers, their main focus. The university tries to make DCs greener or in other words tries to improve energy management and thereby uses less energy. The Papillon Master Client project was developed to estimate the energy consumption of servers being monitored in a DC and to also produce reports. The energy consumption of a server is calculated on the master without the use of additional hardware. An increasingly popular tool in DCs is virtualization. This is used among other things to efficiently utilize the existing hardware. Virtual Machines (VMs) were not supported by the Papillon system.

This work deals with determining the energy consumption of VMs without using additional hardware. This work begins with an analysis of current literature on the subject, a detailed description of the Papillon system followed by the design and implementation phase for testing the newly developed system.

## Acknowledgments

This master thesis was written in 2015 for the Institute for Technical Informatics, Graz, University of Technology in cooperation with University College Dublin (UCD).

I am very thankful for all of the advice and help offered by the Institute for Technical Informatics of Graz, University of Technology and the University College Dublin (UCD). In particular, I want to address my deepest thanks to my supervisors Christian Steger, Damian Dalton, Abhay Vadher and Mario Polaschegg.

A special thanks to my girlfriend Silvia Farkas for all the help and support abroad as well as in Graz. I also wanted to address my deepest thanks to my family especially my dad Peter with girlfriend Ingrid, my sister Cornelia and my brother Jürgen for the support during my studies in Graz or abroad. Last but not least I wanted to address my thanks to my friends for their support and appreciation during my studies.

Graz, August 2015

Andreas Abraham

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Motivation . . . . .	1
1.2	Goal . . . . .	2
1.3	Outline . . . . .	3
<b>2</b>	<b>Fundamentals</b>	<b>4</b>
2.1	Cloud Computing . . . . .	4
2.1.1	Energy Consumption from Data Centers (DCs) . . . . .	5
2.1.2	Metrics, Measurements and Standards . . . . .	6
2.1.3	Carbon Footprint . . . . .	9
2.2	Green Information Technology (IT) . . . . .	9
2.3	Virtualization . . . . .	10
2.3.1	Virtualization Techniques . . . . .	11
<b>3</b>	<b>Related Work</b>	<b>14</b>
3.1	VM Power Metering and Provisioning . . . . .	14
3.2	A Comparison of High-Level Full-System Power Models . . . . .	15
3.3	An Online Power Metering Model for Cloud Environment . . . . .	16
3.4	The Papillon System . . . . .	17
3.4.1	Papillon Architecture . . . . .	17
3.4.2	Papillon System Operation and Work Flow . . . . .	20
3.4.3	Power Model Overview and Generation Process . . . . .	20
3.4.4	Papillon Data Center Power Reports . . . . .	25
3.5	Papillon System in Comparison with Related Work . . . . .	25
<b>4</b>	<b>Design</b>	<b>26</b>
4.1	Overview of the Current Papillon System . . . . .	26
4.1.1	Problems and Issues with the VM Support . . . . .	27
4.2	Requirements of the Components . . . . .	28
4.2.1	Master Requirements . . . . .	28
4.2.2	Client Requirements . . . . .	29
4.2.3	Power Model Creation Requirements . . . . .	29

4.3	Design of the Components . . . . .	30
4.3.1	Design Papillon Master . . . . .	30
4.3.2	Design Papillon Database . . . . .	30
4.3.3	Design Algorithm for VMs . . . . .	31
4.3.4	Design Papillon Client . . . . .	31
4.3.5	Papillon Client Start Routine . . . . .	32
4.3.6	Papillon Client Monitoring and Sending Process . . . . .	34
4.3.7	Design Power Model Creation Process . . . . .	34
4.3.8	Design Client Installer . . . . .	36
4.4	Used Tools and Environment . . . . .	36
4.4.1	Programming Languages . . . . .	36
4.4.2	Apache Tomcat . . . . .	37
4.4.3	System Information . . . . .	38
4.4.4	Yet Another Java Service Wrapper (YAJSW) . . . . .	39
4.4.5	Netbeans integrated development environment (IDE) . . . . .	40
4.4.6	JUnit . . . . .	41
4.4.7	Representational State Transfer (REST) . . . . .	41
<b>5</b>	<b>Implementation</b>	<b>43</b>
5.1	Papillon Master Implementation . . . . .	43
5.1.1	Papillon Database implementation . . . . .	43
5.1.2	Papillon Master Internal Graphical User Interface (GUI) Implementation . . . . .	44
5.1.3	Papillon Data Center Import via Extensible Markup Language (XML) File . . . . .	47
5.1.4	Implementation of the new Power Estimation Algorithm . . . . .	48
5.2	Papillon Client Implementation . . . . .	50
5.2.1	Papillon Client Implementation Overview . . . . .	50
5.3	Changes in the Papillon Power Model Generation . . . . .	53
5.4	Papillon Client Installer for Windows Implementation . . . . .	54
<b>6</b>	<b>Tests and Experimental Results</b>	<b>56</b>
6.1	Overview . . . . .	56
6.2	Verification of the Central Processing Unit (CPU) related Consumption from Virtual Machines and their Hosts . . . . .	57
6.2.1	Hardware and Software Setup . . . . .	58
6.2.2	Test Scenario . . . . .	58
6.3	JUnit Tests . . . . .	60
6.4	Performance Impact Evaluation of the Papillon Client on different Operating Systems . . . . .	61
6.4.1	Overview . . . . .	61
6.4.2	Profiling the Papillon Client using NetBeans Profiler . . . . .	61

6.4.3	Results and Evaluation . . . . .	61
6.5	Verifying the Accuracy of the Estimated Power Consumption of the Pa- pillon System on a Virtual Environment . . . . .	62
6.5.1	Overview of the Accuracy Verifying Process . . . . .	62
6.5.2	Used Hardware . . . . .	63
6.5.3	Used Software . . . . .	64
6.5.4	Problems . . . . .	64
6.5.5	Test Scenarios . . . . .	65
6.5.6	Test Results . . . . .	67
6.6	Papillon Master Stress Test . . . . .	72
6.6.1	Papillon Master Stress Test Setup . . . . .	72
6.6.2	Papillon Master Stress Test Results . . . . .	73
6.7	Import Data Center via XML File . . . . .	74
6.7.1	Conclusion of the Import Data Center via XML File Test . . . . .	74
6.8	Tool to identify the Number of Possible Virtual Machines for Given Server Setup . . . . .	75
<b>7</b>	<b>Conclusion and Future Work</b>	<b>77</b>
7.1	Conclusion . . . . .	77
7.2	Future Work . . . . .	77
<b>A</b>	<b>Visualization of the NetBeans Profiler Results</b>	<b>81</b>
A.1	Description . . . . .	81
<b>B</b>	<b>Client Installer Scripts Source Code</b>	<b>85</b>
	<b>Bibliography</b>	<b>88</b>



# List of Figures

2.1	Mobile-cellular subscriptions, total and per 100 inhabitants, 2005 - 2014 <sup>1</sup> [ICT14] . . . . .	5
2.2	Electricity Consumption for the Year 2007 in billion kWh [GPH12] . . . . .	6
2.3	Power Usage Effectiveness (PUE) . . . . .	7
2.4	Architecture of Native Hypervisor and Hosted Hypervisor . . . . .	11
3.1	Papillon System General Operation and Architecture Overview. This Figure shows a Couple of Server monitored by a Master Server in Real Time [Vad11] . . . . .	18
3.2	Papillon Architecture Diagram from the Master and Client Solution with the various Layers which are Integrated into the Solution . . . . .	19
3.3	Papillon Master and Client Structure with Associated Interfaces . . . . .	20
3.4	Papillon Architecture Data Flow Diagram [Vad11] . . . . .	21
3.5	Device Under Test (DUT) with Power Meter during the Power Model Generation Phase . . . . .	22
3.6	The Power Model Generation Flow Diagram with its different Phases [Vad14] . . . . .	24
3.7	Internal GUI with the Energy Audit Report generation Field . . . . .	25
4.1	The Current Papillon Database Host Table . . . . .	30
4.2	Sequence Diagram showing the Papillon Client Start Routine . . . . .	33
4.3	Client Monitoring and Sending Process Sequence Diagram . . . . .	35
4.4	Sample Output using the Oshi Library [OSH14] . . . . .	38
5.1	The New Papillon Database Host Table Implementation . . . . .	44
5.2	Papillon Internal GUI on the Host Tab from a Data Center . . . . .	45
5.3	Papillon Internal GUI Pop Up Window for Adding a New Host Displaying the Newly Implemented Fields . . . . .	46
5.4	Design of the new Algorithm and its Work Flow . . . . .	49
5.5	Papillon Internal GUI showing Pie Chart for a Sample Point . . . . .	50
5.6	Papillon Client Class Diagram with the Main Parts of the Client Implementation . . . . .	51

6.1	Example Test Setup with three Virtual Machines, a Power Meter and a Test Computer . . . . .	63
6.2	Yokogawa WT210 Power Meter [COR14c] . . . . .	63
6.3	Measurement Error if Clients and Power Meter not Time Synchronous . .	65
6.4	Results of Test Scenario I. Table 6.5 . . . . .	68
6.5	Results of Test Scenario II. Table 6.5 . . . . .	68
6.6	Results of Test Scenario III. Table 6.5 . . . . .	69
6.7	Results of Test Scenario IV. Table 6.5 . . . . .	69
6.8	Results of Test Scenario V. Table 6.5 . . . . .	70
6.9	Results of Test Scenario VI. Table 6.5 . . . . .	70
6.10	Results of Test Scenario VII. Table 6.5 . . . . .	71
6.11	Results of Test Scenario VIII. Table 6.5 . . . . .	71
6.12	Import Data Center with Internal Papillon GUI . . . . .	74
6.13	Tool to get the Maximum Number of Possible VMs on a Physical Server	76
A.1	Graph showing the live memory consumption given by NetBeans Profiler Tool running on Windows 8.1 64 bit . . . . .	83
A.2	Graph showing the live memory consumption given by NetBeans Profiler Tool running on Windows 7 64 bit . . . . .	83
A.3	Graph showing the live memory consumption given by NetBeans Profiler Tool running on Ubuntu 14.04 64 bit . . . . .	84
A.4	Graph showing the live memory consumption given by NetBeans Profiler Tool running on Mint 17 Linux 64 bit . . . . .	84

# List of Tables

4.1	Comparison between YAJSW, Java Service Wrapper (JSW), Apache Commons Daemons (ACD) and Launch4j (L4J) frameworks for wrapping Java Applications [sou14] . . . . .	40
6.1	Table with Results using VMware Player . . . . .	59
6.2	Table with Results using VMware Workstation . . . . .	59
6.3	Table with Results using Oracle VirtualBox . . . . .	60
6.4	Table with Client CPU and Random Access Memory (RAM) Usage on different Operating Systems, Results from NetBeans Profiler . . . . .	61
6.5	Table with used Scenarios together with the Scenario Configuration . . .	66
6.6	Results of the different Test Scenarios with Average Error and Figures . .	67
6.7	Table with the first part of the Stress Test Results from Server I and Server II . . . . .	73
6.8	Table with the second part of the Stress Test Results from Server I and Server II . . . . .	73

# Chapter 1

## Introduction

### 1.1 Motivation

The recent trend is an increase in cloud computing which has resulted in an increasing number of Data Centers (DCs). These DCs are using a lot of energy which has a huge carbon footprint. Energy consumption and the carbon footprint becomes more and more important. There are different options to reduce the carbon footprint of a Data Center (DC). Using sustainable energy is one opportunity. Another opportunity is to reduce the energy consumption. This decreases the carbon footprint and also running costs for the DC.

Physical servers are using a huge amount of energy even when they are running idle. Switching off or suspending the idle running server in a DC immediately decreases the power consumption and also the energy costs of the DC. Another way to save energy is to consolidate applications onto the Virtual Machine (VM), from running on different servers, to one server. It is much better to use the whole capacity of one server.

An accurate soft virtual meter installed on a server allows for accurate power analysis of the server including the power analysis of the applications running on the server. University College Dublin (UCD) was developing a tool which allows the customer to do this. The Profiling and Auditing of Power Information in Large and Local Organizational Networks (PAPILLON) system consists of a Master-Client system which enables the customer to have an overview of the consumed energy in the DC. Furthermore, it is possible to generate reports which details how the energy saving potential in the monitored DC can be completed.

The Papillon system is presently only available for physical servers but virtualization is becoming more and more popular and for this reason, virtualized environments should also be supported using Papillon.

## 1.2 Goal

The Papillon system does not support Virtual Machines (VMs) yet. VMs are becoming more important for DCs because virtualization of servers make it possible to use the given hardware capacity more efficiently. The Papillon system is a Master-Client system which enables the profiling and monitoring of the consumed server energy without the use of a physical power meter. This enables the analysis of the server power usage to be optimized. If a server is running in idle it is not utilizing its energy resource efficiently. When there is only low activity on a server it is the idle power part of the consumed energy that is the dominant one. It is for this reason better to use VMs, so that a few virtual servers can run on a physical server and use the hardware more efficiently.

University College Dublin (UCD) has patented and developed an accurate virtual metering solution that will allow DCs to be more green and help them to save energy which further decreases  $CO_2$  emissions. The university has developed a system, this is a Papillon Master-Client system, which is developed to monitor and observe energy consumed in a server in one or more DCs. However, presently there is no support for VMs. The goal of this thesis is described as follows:

- Complete research on how to enable the VM support for the existing Papillon Client-Master system. The research will consist of developing an algorithm on how it is possible to calculate the power of VMs. Research and review the state of the art for estimating power consumption of VMs or physical servers.
- Become familiar with the existing Papillon system, how it functions including the source code.
- Design the implementation of the VM support for the existing Papillon system and design test scenarios for the source code and the whole extended Papillon system.
- Implement the solution in the existing system on both sides, the Papillon Master and also the Papillon Client.
- Test the extended Papillon system. This testing includes JUnit tests for the Papillon Master together with JUnit tests for the Papillon Client. To augment the testing a verification stage will determine the accuracy of the newly implemented Papillon system. This verification will test the Papillon Master and also benchmark the maximum number of Papillon Clients that can be monitored by one Papillon Master.
- Report test results, evaluation results, known bugs and also future work.

### 1.3 Outline

An outline for this work is described as follows:

- Chapter 2 *Fundamentals* describes the fundamental knowledge and basic background information, which makes it easier to understand the work and its goal. Furthermore, the problem of the increasing energy consumption and other or associated solutions are also explained.
- The related work on this topic including the state of the art of University College Dublin (UCD) is described and explained in detail in Chapter 3 *Related Work*.
- The design of the extended Papillon system including requirements are described in Chapter 4 *Design*.
- In Chapter 5 *Implementation*, the implementation of the new parts are described in addition to the implementation of the extended parts. Additional implementations are also described in this chapter.
- The Chapter 6 *Tests and Experimental Results* describe the tests which were done during different development phases and the tests which were done for different parts of the system. In addition to this, all tests are described, the test results are shown and the evaluation of the test results is discussed.
- Last but not least, Chapter 7 *Conclusion and Future Work* provides a summary of the work completed and also details ideas for future work in this field.

# Chapter 2

## Fundamentals

This chapter describes fundamental and basic background knowledge in detail. It also describes all used or important terminology.

There are different reasons for the increasing demand of energy on the Information Technology (IT) industry. Globalization and rising competitive constraints are two of these reasons. Almost all companies nowadays are using information and communication technology in their business processes. It is very important for companies to react fast when it comes to vital changes and customer needs. In order to make this possible, organizations are using efficient process and cost structures.

Over the last couple of years the IT industry has started to contribute to increasing environmental protection and have been labeling it as "Green IT". The main topics of Green IT include sustainability in the IT industry and the reduction of  $CO_2$  emissions which are created from energy generation.

The main goal of University College Dublin (UCD) is to make IT more "green". The Profiling and Auditing of Power Information in Large and Local Organizational Networks (PAPILLON) system was developed for this very same reason. The Papillon system is a monitoring tool which allows us to determine the actual energy consumption of servers in a DC.

This chapter describes the fundamental concepts of the topic of this thesis including all of the basic terms used. This chapter also includes an overview and detailed description of the Papillon system and its architecture.

### 2.1 Cloud Computing

Cloud computing is becoming more and more important nowadays and the number of DCs are also rapidly increasing. The growing number of e-commerce websites, search

engines, social networks and other business applications (not to mention the rapidly increasing number of mobile phones) are the main reasons for the increase in DC usage. The number of mobile phones was almost equivalent to the number of people living on the earth by the end of 2013 [ICT14]. Figure 2.1 displays the mobile phones and inhabitants all over the world between 2005 and 2014<sup>1</sup>.

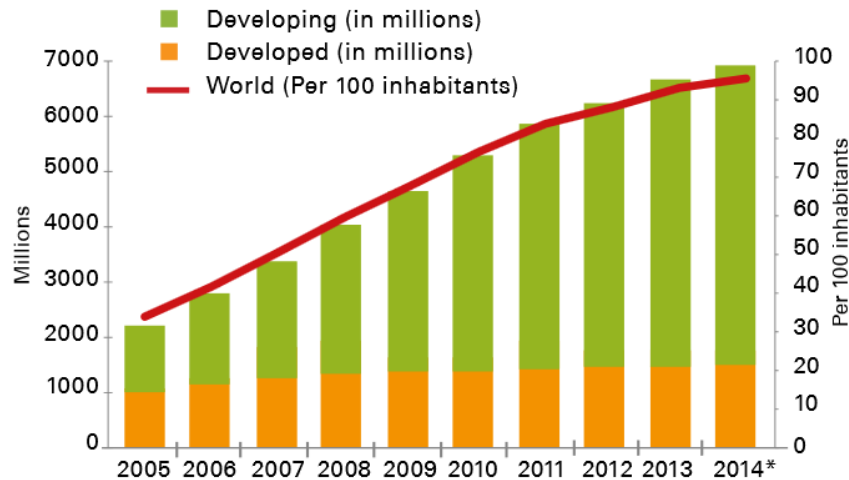


Figure 2.1: Mobile-cellular subscriptions, total and per 100 inhabitants, 2005 - 2014<sup>1</sup> [ICT14]

### 2.1.1 Energy Consumption from Data Centers (DCs)

The energy usage from DCs all over the world has continuously increased in the last years and this trend will continue. In 2010, the worldwide energy consumption derived from DCs is approximately 1.1% to 1.5% of the total world electricity usage [JGK11].

This becomes an important issue because of the carbon footprint and also because of the inefficient use of energy in DCs.

Figure 2.2 shows the amount of used electricity of different countries including the amount of energy which is used for DCs in the world.

<sup>1</sup>2014 Estimate; Source: ITU World Telecommunication/ICT Indicators database



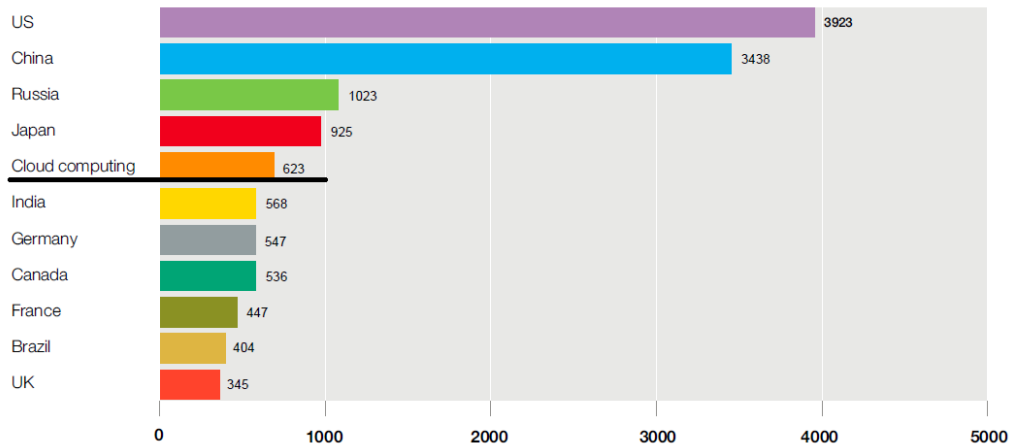


Figure 2.2: Electricity Consumption for the Year 2007 in billion kWh [GPH12]

### 2.1.2 Metrics, Measurements and Standards

To measure DC efficiency, various metrics have been introduced. The most important units are shown and described as follows:

- **Power Usage Effectiveness (PUE)** is used to measure how efficient a DC uses energy in relation to energy effort for hardware, cooling and other things. This value was developed by the Green Grid <sup>2</sup> organization.

A PUE value of 1.0 would be the best value for a DC but this is just theoretical because the value of 1.0 means all the energy which is going into the DC is solely used for the servers and no energy is used for cooling, lighting and other energy consuming components. This is technically not a possible value to achieve, thus, DCs should aim to reach the PUE value of 2.0.

Most DCs all over the world have a PUE value ranging between 2.5 and 3.5 but there is also a DC which has a PUE value of less than 1.1 [Gri14]. Figure 2.3 shows how to calculate the PUE value and the basic concept of this value.

---

<sup>2</sup>The Green Grid organization is a non-profit, open industry consortium of end users and utility companies, to improve the resources and efficiency of information technology and DCs all over the world [Gri14].

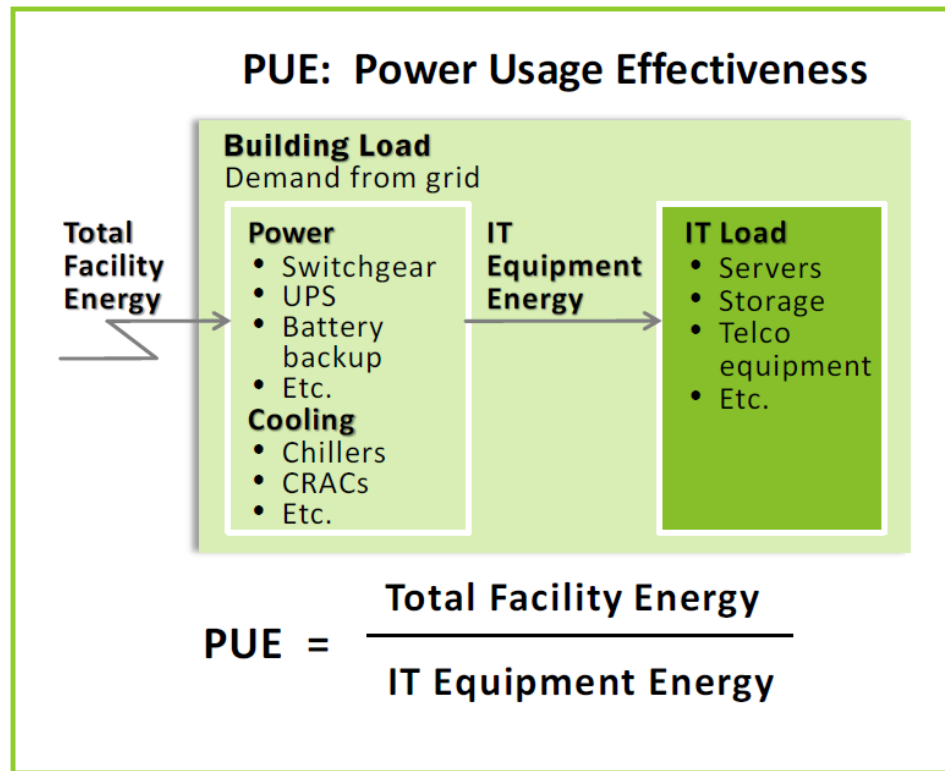


Figure 2.3: Power Usage Effectiveness (PUE)

The major problem with this PUE value is that there are many ways to interpret it and hence it becomes an issue. This mainly has to do with the interpretation of what part is defined for the facility and what part for equipment. The exact definition is as follows:

- **IT Equipment Energy** is the sum of the energy which is used by the IT and supplemental equipment (e.g. computer, storage, network equipment, workstations and monitors) [GGP12].
- **Total Facility Energy** includes all IT equipment energy including additional hardware, which are needed to support the IT equipment. These elements are identified as follows:
  - \* Power deliver components, UPS systems, switchgear, generators, Power Distribution Units (PDUs) and batteries [GGP12]
  - \* Cooling system components, chillers, cooling towers, pumps, Computer Room Air Handling Units (CRAHs), Computer Room Air Conditioning Units (CRACs) and direct expansion air handler units [GGP12]
  - \* Other miscellaneous component loads, such as DC lighting [GGP12]

- **Data Center Infrastructure Management (DCIM)** is the reciprocal PUE value. The exact equation is shown below:

$$DCIE = \frac{1}{PUE} = \frac{ITEquipmentPower}{TotalFacilityPower * 100\%} \quad (2.1)$$

- **DCIM** requirements and functionality are not standardized. The definition depends on the company or the research establishment. DCIM is not a new term in DC management, it includes the software, hardware, telephony, infrastructure and administration [DCI14]. Four of the most popular definitions are described as follows:

- **451 Research**

*A DCIM system collects and manages information about a facility's assets, resource use and operational status. This information is then distributed, integrated, analyzed and applied in ways that help managers meet business and service-oriented goals and optimize a DC's performance [DCI14].*

- **Gartner**

*DCIM tools monitor, measure, manage and/or control DC utilization and energy consumption of all IT-related equipment (such as servers, storage and network switches) and facility infrastructure components (such as Power Distribution Units (PDUs) and Computer Room Air Conditioning Units (CRACs)) [DCI14].*

- **Forrester Research**

*DCIM is a comprehensive approach to managing the physical aspects of the DC. DCIM is the convergence of previous generations of purely facilities-oriented power management, physical asset management, network management, and financial management and planning solutions for DCs. If used properly, DCIM solutions can help I&O professionals address steadily ratcheting pressures to meet business SLAs, lower costs, and improve resource and energy efficiency and long-term facilities planning [DCI14].*

- **Searchdatacenter**

*DCIM is the convergence of IT and building facilities functions within an organization. The goal of a DCIM initiative is to provide administrators with a holistic view of a DC's performance so that energy, equipment and floor space are used as efficiently as possible [DCI14].*

### 2.1.3 Carbon Footprint

In the last years, the term Carbon Footprint (CF) has become increasingly popular in the world. The exact definition of CF is as follows:

*"The Carbon Footprint is a measure of the exclusive total amount of carbon dioxide emissions that is directly and indirectly caused by an activity or is accumulated over the life stages of a product [WM07]."*

It is important to note that it is not only carbon dioxide which has potential for climate warming. Also other gases like Methane have a potential for climate warming but it is not included in the carbon footprint. It could easily be included in the carbon footprint but this would not make the carbon footprint more accurate. Either just one gas is used or all gases which can have an effect on climate warming. In this case only the most potential gas is included in the term CF.

## 2.2 Green IT

The amount of energy which is used for IT, especially for DCs, has increased rapidly. This becomes an issue in the world and many organizations have been making this topic an important target for reduction on their task lists. There are different ideas and ways to reduce energy in IT. Green IT means the efficiency of components like the server, the cooling for the server and the router, and new research and tools is looking at providing energy efficient architecture and infrastructure for DCs.

In older DCs, where growth is achieved on demand, it is often constructed without the planning of the cooling and air system. This then becomes a big issue because the temperature increases and the costs for cooling explodes. Another important point is that the server hardware is less efficient if the temperature is too high and hardware failures can also occur if the temperature is too high. The architecture of a server room should be designed to allow the air of the server racks to easily circulate. The costs for cooling is often the same as the costs for providing the energy for the server usage. The temperature within a DC is an important parameter.

IT industries start their energy-conscious initiatives with the label called "Green IT". The main aim of Green IT is to reduce  $CO_2$ <sup>3</sup> emissions, make hardware more efficient and to also look for better usage of software. In older DCs, 18 degrees Celsius was the default temperature in the facility but a study showed that this is too low and expensive to maintain. It is much more efficient to work with 26 degrees Celsius. The hardware manufacturer develops the new hardware to use less energy and work more efficiently [Lam10].

---

<sup>3</sup>Carbon Dioxide is colorless and odorless gas which is one of the biggest reasons for the climate change. It can be produced in many different but one of the biggest parts is the energy production.

The DCs that use increasingly renewable energy from wind turbines or solar panels also reduce the costs and the carbon footprint and thus use more green energy [CSK09].

A good example for the usage of renewable energy is the international company Facebook. Facebook is a social media website which has more than 800 million users all over the world and is thus the largest social network in the world. Facebook was the first company which took the major step to construct a DC which uses excessive renewable energy [GPH12].

Although many servers in a DC only use a small amount of their capacity, this still requires a great deal of energy. Depending on the server type, many servers will consume half of the maximum energy in idle. For a server with little activity, the energy consumption of the server is high. It would be better if the server would use more capacity and thus be more energy efficient. A way to use the server hardware more efficiently is to use virtualization. If a server is running long in idle it is better to switch it on and off to save energy and also cooling costs.

## 2.3 Virtualization

The basic idea of virtualization is to run multiple VMs on a single computer. These VMs run simultaneously and independently using the hardware of a single physical computer. Each of these VMs start with a specified Operating System (OS) and act like a self contained computer. The improvement of the virtualization technology is explained below:

**Using Hardware more Efficiently:** Virtualization allows multiple VMs to run on one physical server which uses the existing hardware more efficiently instead of using more physical servers.

**Multiple Operating Systems and Applications Simultaneous on a Server:**

This technology allows different OSs or applications to run on one physical server at the same time<sup>4</sup>.

**Saving Costs:** If VMs are consolidated on less physical machines the costs for electricity and hardware maintenance decrease.

**Encapsulating of Applications:** Each VM is isolated or in other words encapsulated. When an application is hacked or infected by a computer virus the other VMs on the server are not affected.

Not only are there advantages to virtualization technology but there are also disadvantages such as for example five different VMs running on a server that has a hardware problem. This means all of these VMs are affected.

---

<sup>4</sup>Only hypervisor based virtualization technologies support multiple OSs.

### 2.3.1 Virtualization Techniques

Virtualization technology is divided into two different virtualization technologies where one is the hypervisor based technology and the other one is the container based technology. The Hypervisor is also known as Virtual Machine Monitor (VMM).

Container based virtualization is also known as system level virtualization. This is a special kind of OS which can be used like a normal OS. The big difference to a standard OS is, that this OS allows users to create VMs and run this encapsulated on the OS. The encapsulated OS called container is a VM isolated from the host OS. An advantage of this system is the performance is very efficient, but the disadvantage is that the server and the container OS have to be the same [Li10].

The hypervisor based virtualization technology is divided into two different types. The first type is the native hypervisor and the second one is the hosted hypervisor. Figure 2.4 shows the different hypervisor types which are described as follows:

**Native Hypervisor:** This type of hypervisor is a software layer which is directly installed on the physical hardware instead of an OS. In this case the hypervisor controls the hardware. The native hypervisor is also known as bare metal hypervisor. Each VM is running as a process on the host.

**Hosted Hypervisor:** This type of hypervisor is installed on a standard OS such as on Windows or Linux. The hypervisor is running as an application on the OS. The hardware is in this case controlled by the OS and not the hypervisor.

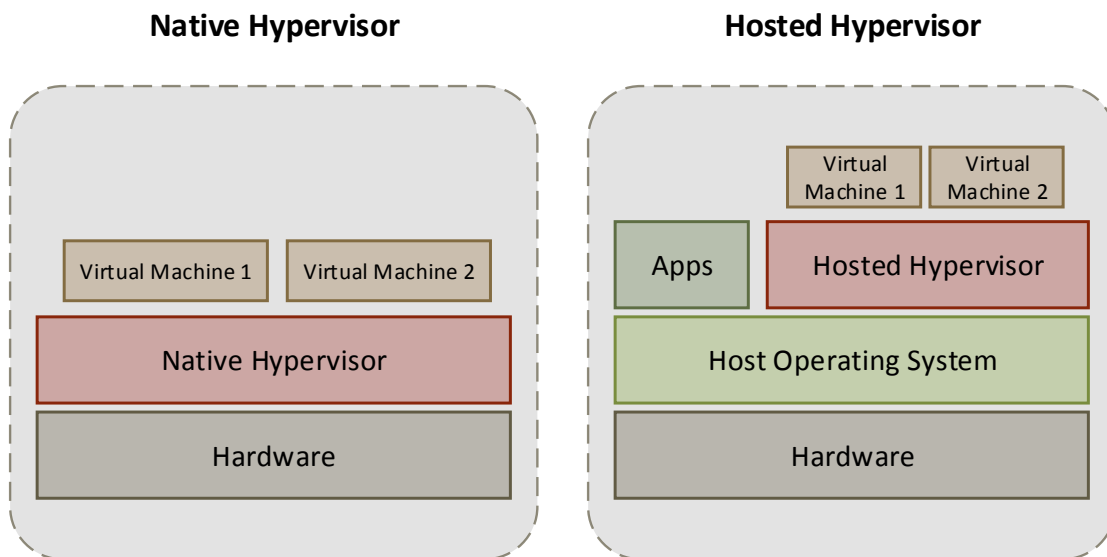


Figure 2.4: Architecture of Native Hypervisor and Hosted Hypervisor

One big advantage of the hypervisor based virtualization is that multiple guest OSs are supported. The native hypervisor has a better performance compared to the hosted hypervisor because it is sitting directly on the hardware.

- Examples for native hypervisor are VMware ESXi<sup>5</sup>, Microsoft Hyper-V<sup>6</sup> and Kernel Based Virtual Machine (KVM)<sup>7</sup>.
- Examples for hosted hypervisor are VMware Player<sup>8</sup>, VMware Workstation<sup>9</sup> and Oracle VirtualBox<sup>10</sup>.

Section 2.3.1 *VMware Virtualization* VMware Virtualization describes the virtualization tools from VMware because their tools are mainly used for this work. Furthermore, Section 2.3.1 *Oracle VirtualBox* describe the virtualization tool from Oracle because this was also applied to this work.

## VMware Virtualization

VMware is the most popular software company for virtualization software which allows customers innovation and optimization of their companies [Inc14b]. The company was founded in the United States in 1988. VMware is the leader of private cloud virtualization [Inc14a]. The company develops virtualization software for different areas of application. A brief list of hypervisor products are described below:

- **Hosted Hypervisor**

**VMware Player:** It is a freeware tool for non-commercial use, which can be installed on Windows and Linux systems. VMware Player only allows one VM to run at a time.

**VMware Workstation:** This virtualization tool is the extended version of the VMware Player. It allows several VMs to run at the same time and also includes extra functionality like taking snapshots of the virtual machine's state.

**VMware Fusion:** This is the virtualization tool for Mac.

---

<sup>5</sup><http://www.vmware.com/products/vsphere-hypervisor>

<sup>6</sup><http://www.microsoft.com/en-us/evalcenter/evaluate-hyper-v-server-2012-r2>

<sup>7</sup><http://www.linux-kvm.org>

<sup>8</sup><http://www.vmware.com/products/player>

<sup>9</sup><http://www.vmware.com/products/workstation>

<sup>10</sup><https://www.virtualbox.org/>

- **Native Hypervisor**

**VMware ESXi:** This is a bare-metal hypervisor, which is installed on the server, and the administration is done with vSphere.

**VMware vSphere:** The new vSphere hypervisor includes the ESXi in its architecture. The new vSphere hypervisor is the new flagship of VMware for data center virtualization. vSphere has several new features such as:

- **VMware vMotion:** This is a feature from vSphere to move VMs between servers or across clusters. The VM is running during the moving process which is called live migration[Inc15].
- **VMware Distributed Resource Scheduler (DRS):** is a feature which decides, after a VM is powered on, which server should be used to run this VM. DRS will place it on a host which matches automation level criteria otherwise a recommendation will be generated to decide which host is the best for this VM[Inc15].
- **VMware vSphere Distributed Power Management (DPM):** in combination with DRS and vMotion allows vSphere to automatically move VMs through the cluster which resulting effect is to save energy [Inc15].

### Oracle VirtualBox

VirtualBox is a free Open Source Software that can be installed on Windows, Linux and Mac machines. It is a hosted hypervisor under the conditions and terms of the GNU<sup>1112</sup> General Public License (GPL) version 2.

VirtualBox allows multiple VMs to run simultaneously with different OSs[Ora15]. Two example features of VirtualBox are described as follows:

**Snapshots:** VirtualBox allows snapshots to be taken of the current VM state. The user is able to revert the machine to the snapshot state at a later point in time.

**Remote Machine Display:** This VirtualBox feature allows high performance access to the running VMs.

---

<sup>11</sup>GNU is Not Unix (GNU)

<sup>12</sup><https://www.gnu.org>



# Chapter 3

## Related Work

### 3.1 VM Power Metering and Provisioning

Virtualization is used to use the server resources more efficiently and this is often used for cloud computing. One huge disadvantage of virtualization is that power monitoring for VMs is not possible. The popularity of data center power management is very much on the rise. In modern data centers the utilized servers have a built-in power meter to measure their power consumption.

The paper VM Power Metering and Provisioning [KZL<sup>+</sup>10] describes the power management for data center. It is a well-known fact that power management in data centers is very important. A modern server can have a built-in power meter but there are other solutions to accurately measure the power consumption of servers. It is already possible to measure the consumed power of servers with the PDU this can measure the consumed power directly on the energy plug. The usage of VMs has many advantages like the safe isolation of co-located workloads, the option to enable multiple workloads on fewer servers, optimized resource usage and the reducing of idle power costs. Details of a solution named "Joulemeter" is introduced and described in this paper. Another important aspect of this paper is the creation process of a power model which is needed to calculate the power consumption. Furthermore, this paper shows the possibility to achieve a high accuracy with a low runtime overhead [KZL<sup>+</sup>10].

The Joulemeter system is used for decreasing the costs for energy in data centers. It has the same functionality as a normal physical power meter. It is possible with the use of power models, to track the energy consumption in VMs including each important hardware resource which will be done with the hypervisor hardware power states. The main idea is to convert a full systems power measurement capability into the VM energy usage. The first step is to start one VM on the server. After this the server power measurement starts. The measured idle power of the server is subtracted from the measured values meaning the leftover is the consumed energy from this VM. The next

step is to redo this for all VMs which are running on this server. Theoretically the energy consumption on all VMs should be the same but in reality it is not. This is due to the different workloads and settings of the VMs and the number of VMs running. To calculate the consumed energy with the use of the power model and the energy consumed with the associated resources. To calculate the energy is the Central Processing Unit (CPU), disk, memory, network cards and graphic cards used as resource [KZL<sup>+</sup>10].

In this paper they talk about power meter challenges which means the VM tracking. The two challenges are described as follows:

- Power measurement at fine time granularity
- Label resource use per VM

The design of the Joulemeter is also described in this paper. The idle power of a server is often more than 50% of the whole energy consumption from the server. The most dominant resources on a server that are important for the power consumption calculation are the CPU, memory and the disk [KZL<sup>+</sup>10].

## 3.2 A Comparison of High-Level Full-System Power Models

To maximize the energy efficiency the data center operators and users have to understand the relationship between the resource usage and the system-level power consumption. This understanding will allow the system optimization on one side and on the other side the shift of workloads to other machines which then allows users to power off unused servers and thus save energy [RRK08].

The paper A Comparison of High-Level Full-System Power Models [RRK08], explains that the accuracy of the system is less than 10% average error. And if a smaller error is required, there are a few optimizations are needed. A model should be accurate enough to enable the energy savings, and the model should be created very quickly. An experiment shows that one second samples are very fast. The developed system should be able to run on as many systems as possible, which includes mobile devices, desktop and server systems, different processor architectures, memory and other storage technologies. Furthermore, it should be easy to create new models for new systems. Last but not least should, everything should be as easy and simple as possible. This work describes how high-level full-system power models are created. Furthermore a comparison of the accuracy of these models for each instance is described as well [RRK08].

In this paper a tool named "Mantis" is used to create the full-system power predictions that are fast and accurate. Mantis requires an AC power meter. The power readings and the software metrics were saved every second. A calibration workload process is

running which is used to gather the measurements used to generate a matrix. These software metrics with the power readings is used to predict the power without using the Alternating Current (AC) power meter. The calibration workload process is using different benchmarks and other programs which are stressing the dominant components of the server, which are the CPU, memory and the disk. The calibration is based on the idle power of the server [RRK08].

For the evaluation and testing part of this paper, five different models are compared. Model one is a simple model which predicts a constant power regardless of resource utilization. Model number two and three are created using the CPU and the fan. The fourth model is proposed by "health". This model is a linear model that is using the CPU and disk Input/Output (I/O). The evaluation of the models was completed by generating the predictions utilization the models and also the measuring of the full system. These values are gathered every second and compared afterwards [RRK08].

The tests have shown, that the model number five is the most accurate model of these five. The error is in a range between 3% and 5%. The model number one was the worst one from these five with an error range from 8% to almost 16% [RRK08].

### 3.3 An Online Power Metering Model for Cloud Environment

The energy consumption produced from large scaled cloud computing, a huge amount of parallel running server became a major operational cost. Decreasing the energy consumption of data centers will not only reduce the energy costs, but rather it will reduce the greenhouse gas emission as well. Therefore, it becomes to an important reason to reduce the power consumption used for cloud computing. One possibility to optimize the power consumption is virtualization. New challenges for the virtualization of data center appears like the missing technique for accurately measuring the power of VMs [LWYG12].

In the paper An Online Power Metering Model for Cloud Environment[LWYG12] explain their idea of an online power metering method for VMs. For the analysis was a linear regression model used. This analysis uses three parameters with the most impact on power consumption of VMs which are the CPU, the disc and the memory. A basic model is created out of analyses of the characteristic model based on the performance. By measuring the real power consumption and compare it with the estimated value, the sub classified piecewise linear model will improve the basic model [LWYG12]. They created for the basic model, based on statistical observation, a percentage of the components accounting dynamic impact on the power consumption. The used components for this power estimation are the CPU, disc and memory and an additional variable  $e$ . The following step to improve the model is to train the basic model. The result will

be the weights  $\alpha$ ,  $\beta$  and  $\gamma$  which should increase the accuracy of the resulted estimated power. Furthermore, the variable  $e$  is an adjusted bias. The conclusion of this paper is that the accuracy is better if there are just one or two machines running on the server. The accuracy decreases according to the number of VMs running on the physical server [LWYG12].

## 3.4 The Papillon System

University College Dublin (UCD) has developed a solution to monitor and observe the energy consumption of a data center including all of the servers in operation. The main reason for this was to save energy and make the data centers more efficient and more "Green". The name of this IT energy management tool is Papillon, which is a Master-Client System. This system should make it possible to monitor a whole data center and later create automatic reports which shows the energy consumption and where are the spots where the data center can save money and energy. This section describes the Papillon system in detail, including an architecture and workflow overview.

### 3.4.1 Papillon Architecture

The Papillon System is a Master-Client system which consists of a master server by what can monitor and observe many clients. These clients are running on several servers and send the system information to the master. In this section the Papillon system with its architecture described in detail as follows. Figure 3.1 shows the general operation overview of the Papillon system.

The Papillon system operation flow is described in the following points below [Vad11].

- First for all new server configurations, a new power model has to be created and verified. An overview of the power model and also the generating and verifying process are delineated in particular in the 3.4.3.
- The power model has to be saved in the power model data base. For every different server hardware configuration has a new power model to be created, which can after it is saved in the database, reused for all same server configurations. If a data center has just one server configuration, means that only one power model is needed for the whole data center. The accuracy of the system can be increased by updating the power model with more power values. This can be done with the power model generation tool.
- All generated and verified power models are saved and configured in the Papillon power model database.

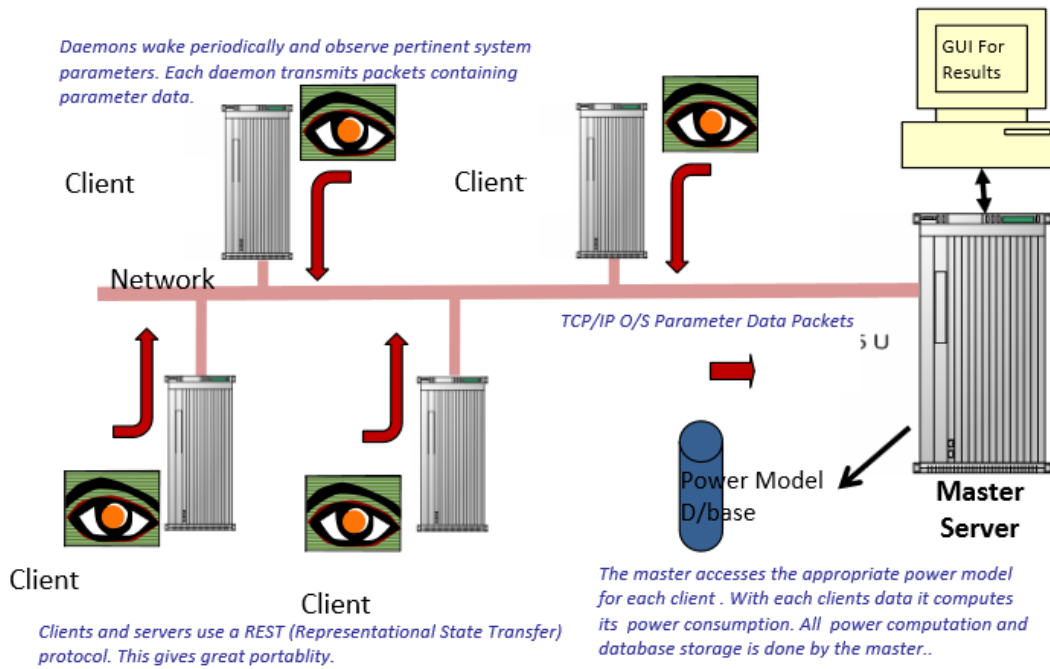


Figure 3.1: Papillon System General Operation and Architecture Overview. This Figure shows a Couple of Server monitored by a Master Server in Real Time [Vad11]

- The Papillon Master is used to run the Papillon monitoring tool and also the Papillon internal Graphical User Interface (GUI). This master server has to use the latest release of the Papillon Master software package. The Papillon Master is used to save and store all power information from all observed clients and also from the master itself. All these information are stored in a Structured Query Language (SQL)-lite database for the duration of the Papillon operation.
- The Papillon Clients have to be loaded with the latest Papillon Client release software package. The clients are used to monitor the system parameter and send this data to the Papillon Master. The master computes the estimated consumed power of the server from the given information of the Papillon Client and with the power model for this server configuration.
- The Papillon Master has an open Application Programming Interface (API) which grants the user access to the power profile results and also for report generation. This API can also be used for 3<sup>rd</sup> party report generation tools.
- The Papillon system can also be used for measuring power of other electronic components within a data center if there is a power model of the specified equipment is generated. An example for other electronic components within a data center are described as follows:
  - Network Switches

- Power Distribution Units (PDUs)
- Network Attached Storages (NASs)

The Papillon system consists of a master and the clients which are using several layers of integration into the Papillon solution. Figure 3.2 the different layers and the Papillon integration in this layer and also that the Papillon system is very flexible in terms of making sure that it will work with different platforms and OSs.

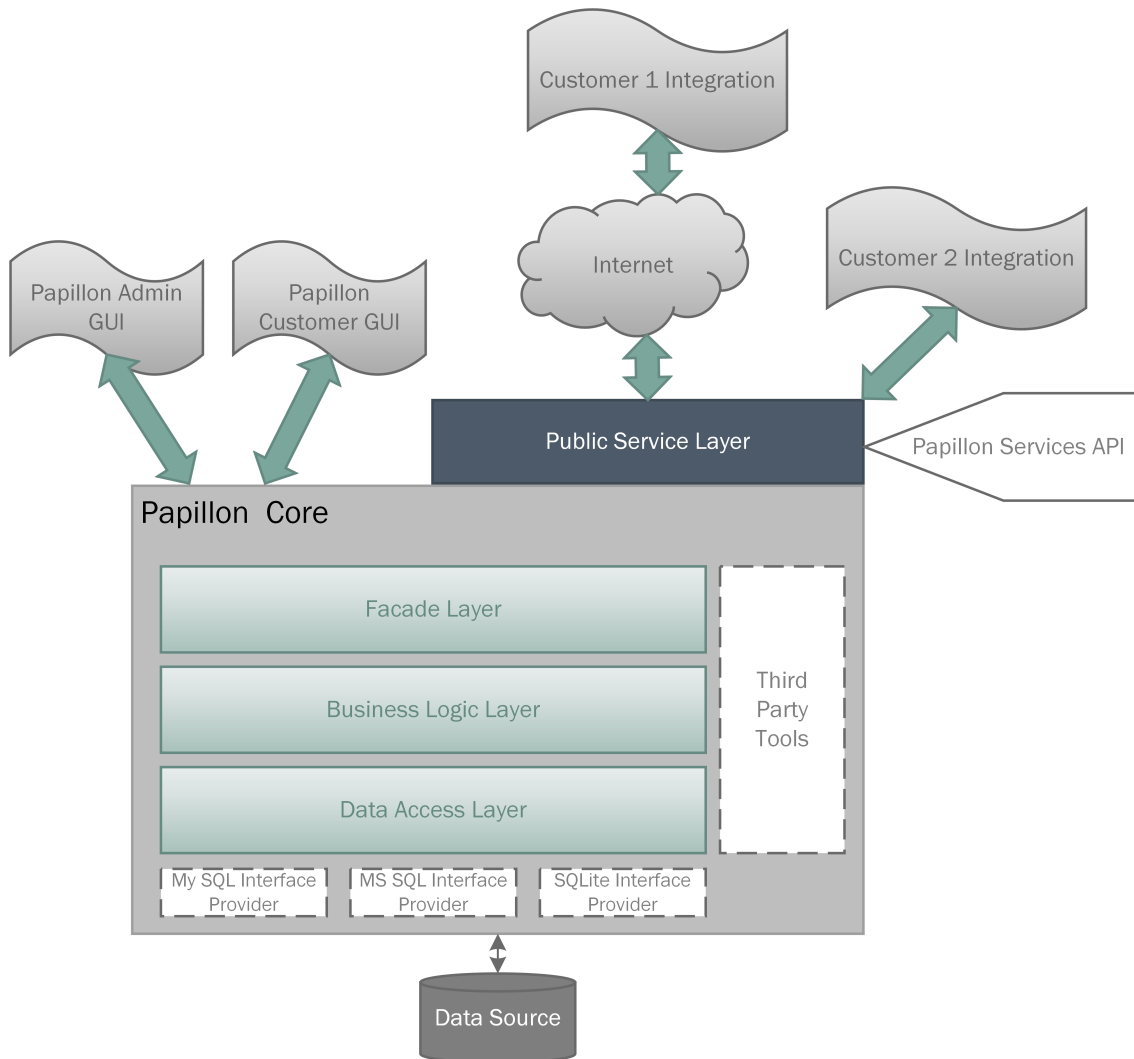


Figure 3.2: Papillon Architecture Diagram from the Master and Client Solution with the various Layers which are Integrated into the Solution

The architecture of the Papillon system is shown in Figure 3.3 and also its organization, structure and interfaces.

The Papillon internal GUI is also known as Papillon management tool details functionality like a heartbeat check of the installed clients (agents), deployments of the agents,

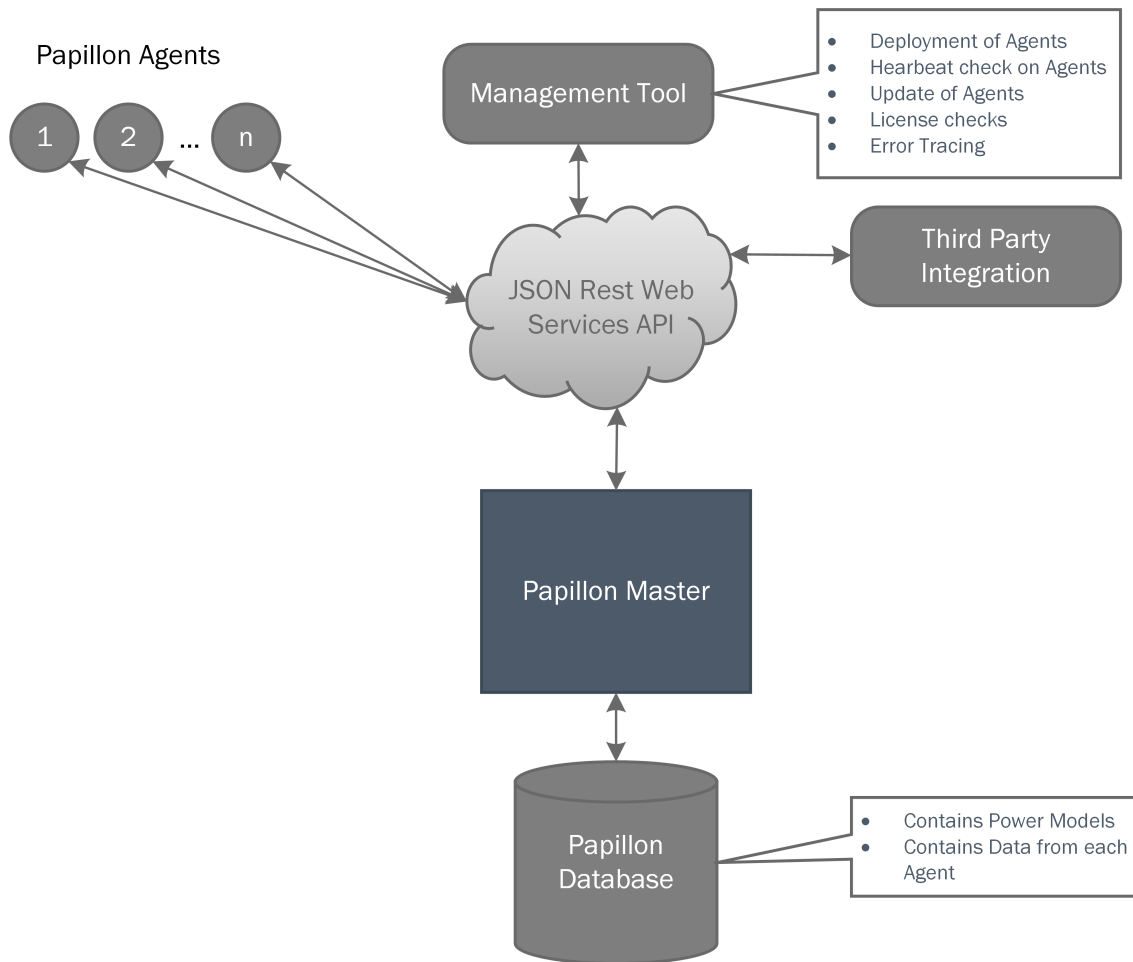


Figure 3.3: Papillon Master and Client Structure with Associated Interfaces

update of agents and associated licensing and error tracing [Vad11].

### 3.4.2 Papillon System Operation and Work Flow

In this section the Papillon systems operation and work flow is described in detail. Figure 3.4 indicates a data flow diagram that details the architecture of the Papillon system and the data flow during operation.

### 3.4.3 Power Model Overview and Generation Process

The central part of the Papillon system is the power model. Each server with different hardware configuration has a unique power model. This power model consists of a number of values which describes the power behavior of the server with specified system parameters. Papillon is measuring these parameters which are applied to calculate the

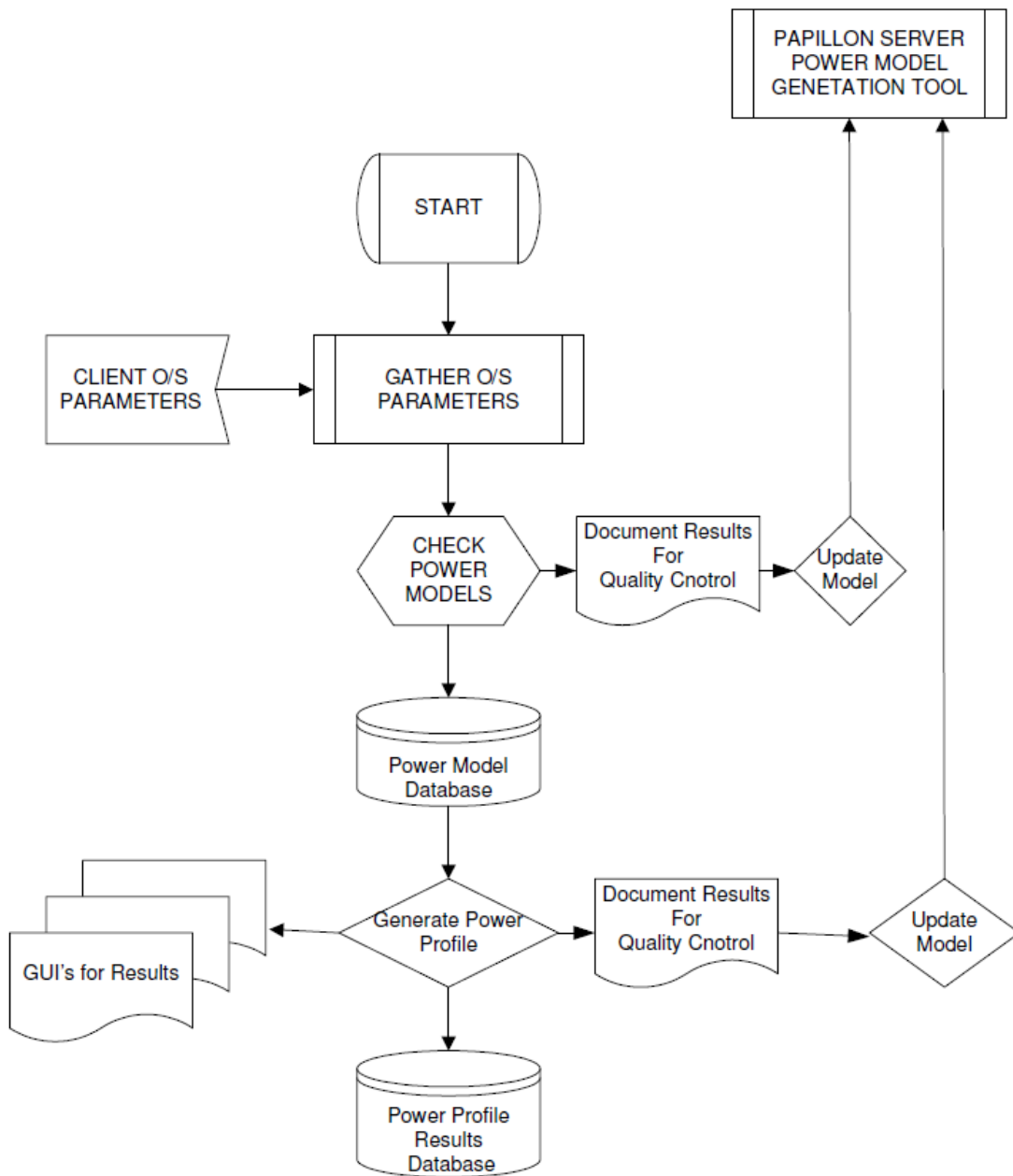


Figure 3.4: Papillon Architecture Data Flow Diagram [Vad11]



consumed power of the server. This used up power can be computed by employing the appropriate power model for each host.

To create a new power model a special work load generator is executed on the server. To record the power profile of a server, a power meter is required. In this case a Yokogawa WT210 power meter is applied for confirmation and also recommended by UCD because of their compatibility of their software. The tested server (Device Under Test (DUT)) has to have an OS installed for executing the profiling and benchmark programs. Figure 3.5 shows the DUT with the power meter during the power model generation phase. All generated data are stored on the server and after successfully power model creation the data are transferred manually to UCD [Vad14].

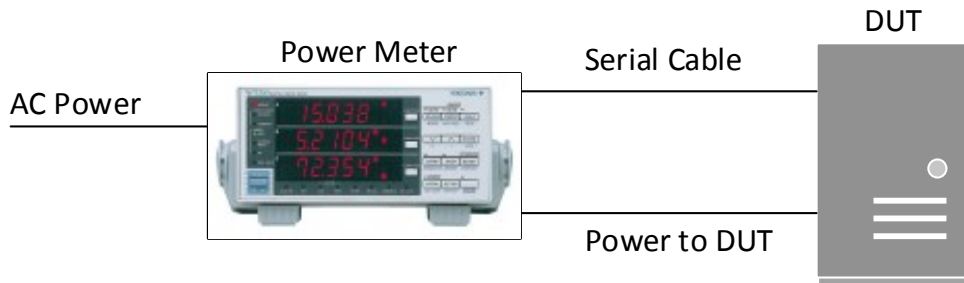


Figure 3.5: Device Under Test (DUT) with Power Meter during the Power Model Generation Phase

One sample point of the power model consists of different values, which are the CPU, Memory I/O, Disk I/O, Network I/O and the measured power value from the power meter. After a power model has been produced its accuracy can be verified. The power model generation process takes six hours, this is necessary to gather enough data samples to sustain an accurate power model from the server.

During the power model generation phase, different benchmarks are running on the DUT to simulate activity on the server. This makes the power model more accurate. The used benchmark is the tiobench [KP14] which is available for Linux machines. The power model generation has to follow the guidelines of UCD. because the idle power is important as well as the power model [Vad14]. Figure 3.6 shows the power model generation flow chart and its different phases. The different phases are described in detail as follows:

- In **Phase 1** the server configuration details of the server and the used OS are defined.
- **Phase 2** focuses on opening a connection from the server to the power meter. It is highly recommended to use Ubuntu as an OS for the power model generation process, also because of the connection to the power meter.

- In **Phase 3** of the power model generation process, the benchmarks are started and checked. Also in the beginning and at the end the server has to be at least twenty minutes in idle because of later idle power calculation. Furthermore, the sampled data are stored on the server, including the measured power values.
- In the final **Phase 4**, the power model file is generated and stored. Now, the power model can be used for this kind of server hardware configuration.

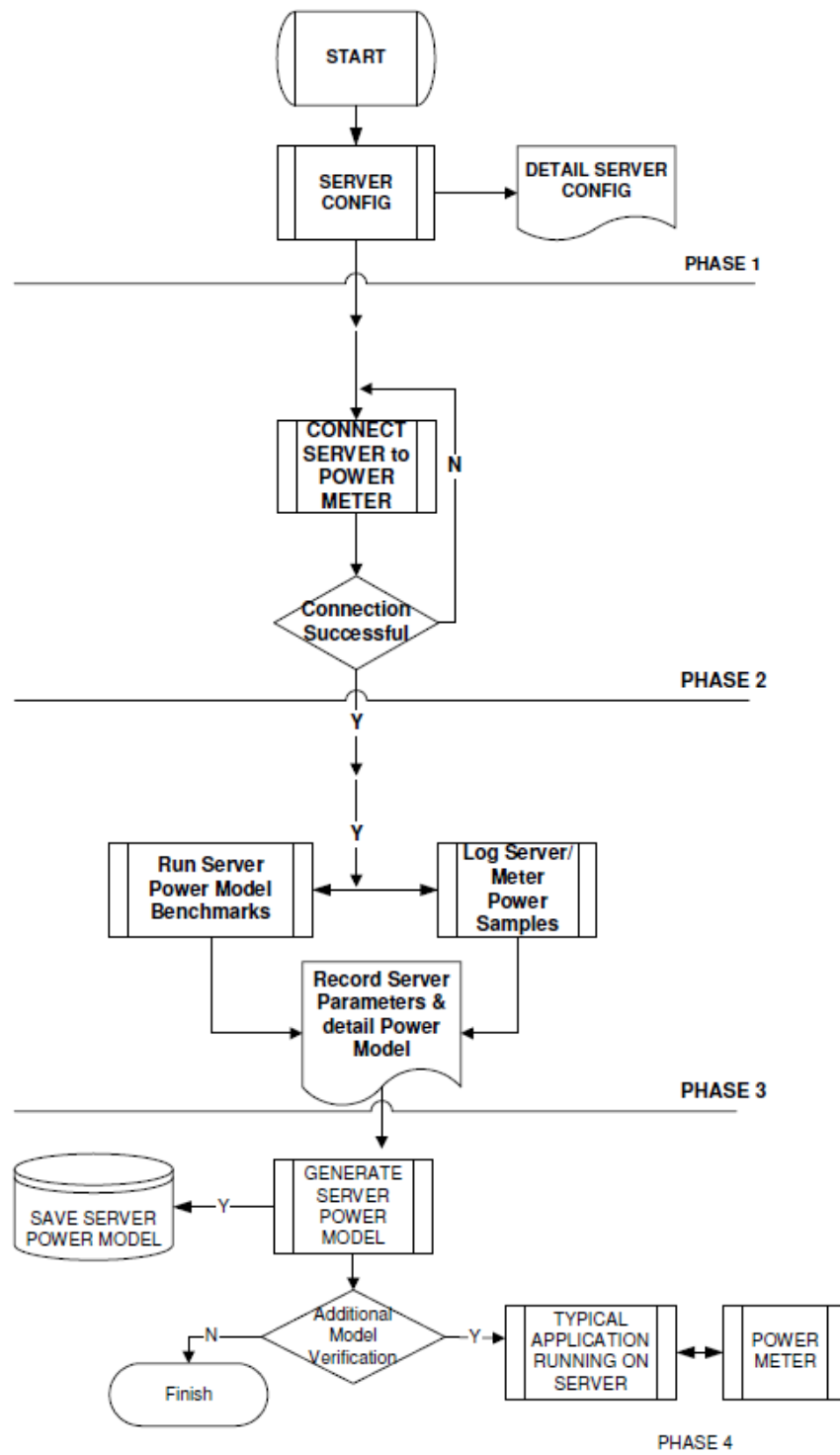


Figure 3.6: The Power Model Generation Flow Diagram with its different Phases [Vad14]

### 3.4.4 Papillon Data Center Power Reports

Papillon allows the costumer, who has this system installed in the data center, to create operational reports which include different data and facts about the data center. The energy audit report can be created the first time after the Papillon system has monitored the server for a period of seven days. The report can then be created easily by pressing the create button shown in Figure 3.7.

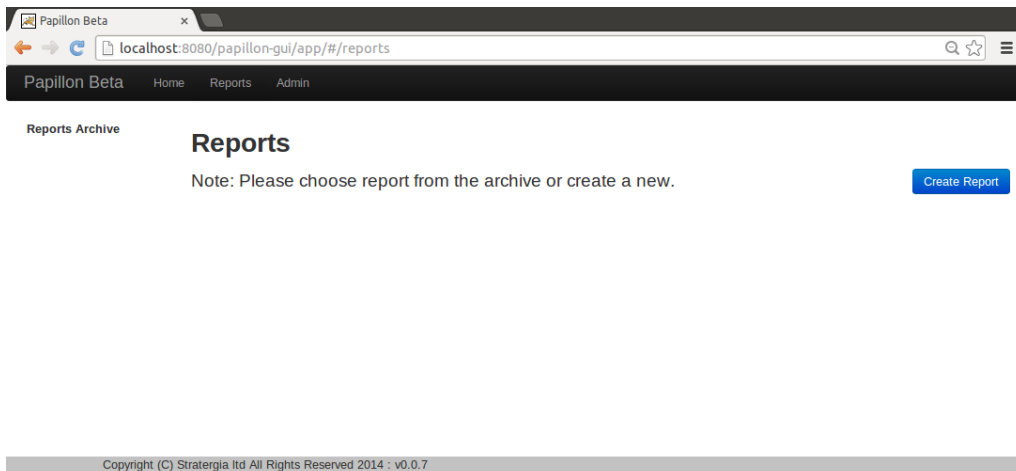


Figure 3.7: Internal GUI with the Energy Audit Report generation Field

The information, which will be in the created energy audit report include the following points described as follows [Ltd15]:

- The carbon footprint as well as the operational costs of every server
- Recommendations and identifications of servers, which are candidates for energy saving improvement
- Detailed analysis of each rack

## 3.5 Papillon System in Comparison with Related Work

Comparing the related works [RRK08, KZL<sup>+</sup>10, LWYG12, et.al.] with Papillon shows that the most significant difference the way the power model is created. UCD for example, is measuring and recording the needed data during the power model creation process every sixty seconds. In related work was one second instead of sixty seconds used to gather the system information.

Furthermore, the used algorithms are different, which are responsible for the power estimation.

# Chapter 4

## Design

In this chapter, the requirements, and the design of the solution is described and explained in detail. Furthermore, the involved parts of the Papillon system and their modifications are identified in detail as follows. The design, described as follows, should make clear that all requirements are included. Moreover, the design is related to the "Keep it Simple" principle.

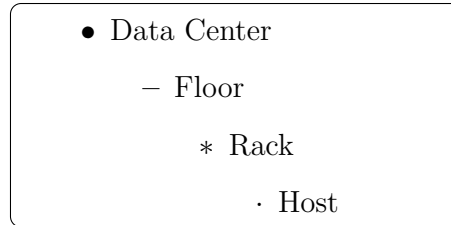
### 4.1 Overview of the Current Papillon System

The actual Papillon systems, allow estimating the consumed power of servers, without using additional hardware. The estimation utilize different system parameter to make an accurate power analysis. The considered system parameters are listed as follows:

- CPU consumption
- Memory I/O
- Disk I/O
- Network I/O

These information parameters are gathered from the whole server, including the three top applications. The three top applications are these applications, which are using the most server capacity. These three applications can change during the life cycle of the system. The power consumption of the whole system as well as from the three top applications is calculated and stored in a database. The Papillon system is developed for whole data centers. For this reason, the system is structured into different layers, which is reflected in the way the database is created and stored.

The structure of a data center hierarchy is represented in the Papillon system and identified as follows:



This Papillon system is developed for monitoring more than one data center with different floors, racks and hosts. This structure reflects the database schema, related to a data center hierarchy.

To display the results of the Papillon system, this includes the consumed energy from the server as well as the consumed energy from the three top applications; an internal GUI is developed and used as a front end. This internal GUI can be utilized from the administrator of this system as easily as from the customers itself. In other words, this front end grants the user to access the power estimation data from a server.

To enable the VM support, so that the consumed power of VMs can be calculated by Papillon, the existing system has to be extended. The goal of the new system is the full virtual environment support. All solutions, which are developed for this case, are following the "Keep it Simple" principle.

#### 4.1.1 Problems and Issues with the VM Support

The Papillon system was designed and developed to estimate the consumed power of a physical server without using any additional hardware. Presently, it has become more and more popular to virtualize parts, or even to virtualize whole data centers. The actual Papillon system is not able to estimate the consumed power of VMs correctly.

The Papillon system is used to accurately estimate the power consumption of a server, utilizing different hardware parameter like CPU usage, memory I/O, disk I/O and network I/O. The problem with the VMs is that it does not have any physical hardware. VMs are using virtual hardware, which provides the hypervisor. The hypervisor get physical system resources from the server and provide these resources for the VM. The VM does not know that the hardware is just virtual not physical.

To obtain a more honest understanding about, how the virtual hardware interacts with the real physical hardware, a couple of tests are required. These tests should make clear how the hypervisor interacts with the physical server and the VM. These tests focus on the CPU consumption, because this is the dominant part of the power consumption. These tests and their results are shown, explained and described in detail in Section 6.2 *Verification of the CPU related Consumption from Virtual Machines and their Hosts*.

A very important aspect that needs to be considered is the idle power of the server. The idea is, if there is more than one VM running on a physical server, and all of the running machines are in idle, they all share the same idle power of the server. This means if we want to calculate the consumed power of a VM, then we have to scale the idle power of the server so that the correct idle power part is used to estimate the consumed power of the virtual server.

## 4.2 Requirements of the Components

### 4.2.1 Master Requirements

The Papillon Master has to be extended, so that it is able to handle the VMs in a different way compared to the physical machines. Therefore, the master needs the information about the current environment. The current environment should be represented by a flag implemented in the Papillon database table. This flag has to be set when a new host is added or created on the Papillon Master. This flag can be displayed in the internal Papillon GUI on the add host site.

The Papillon Master has the functionality implemented which allows it to install a whole data center via Extensible Markup Language (XML) file. For this reason is it necessary to implement the flag also in this installing data center via XML file functionality. This flag defines if the current host is a physical machine or a VM, which can implement as an enumeration. The enumeration named `Environment` consists of two values, which are `PHYSICAL` and `VIRTUAL`. The default value of the `Environment` enumeration should be `PHYSICAL`.

The next flag that should be available, if the current machine is a VM, is a flag that represents the number of VMs running on the physical server. This value is important and necessary for estimating the correct power consumption of the VMs and their applications. If the host is virtual, then also the number of VMs has to be set. This value is not a null value where the default value is one. The flag should be available to set in the internal Papillon GUI if `VIRTUAL` is selected. Otherwise, it should not be possible to change this value. Furthermore, installing a DC via an XML file functionality need to be extended, in the way that the number of VMs running on the server has to be set.

With the knowledge of the host environment, the Papillon Master can decide which algorithm is used for calculating the consumed power. There has to be an "if - else" statement to decide which algorithm to select. If the current host is a physical machine, there should be no changes, which mean that the power estimation should be done same as before. However, if the current host is a VM the Papillon Master will use the new algorithm for estimating the consumed power. The new algorithm consider, that the

host is a VM and the number of VMs running on the server is set, to estimate the consumed power correctly.

The Papillon Master has the power model of the current host. With this power model, the idle power of the server has to be calculated. Therefore, the Papillon Master needs an updated version of the power model to ensure that there is a way to calculate the idle power of the sever. A new method will be implemented in the master which returns the idle power of the server which is used for calculating the amount of consumed power of the VMs.

The consumed power of the applications, which are running in the VM, the master does not have to be changed because the new algorithm calculated the whole amount of consumed energy and the Papillon Master scale this value afterwards to get the consumed power of the applications.

#### **4.2.2 Client Requirements**

There are currently two different Papillon Clients, one is for Linux based OSs and the second is for Windows OSs. Furthermore, the old client consists of an executable, which is excessively large. Moreover, the old client uses more system resources during the runtime process as necessary. Because of this, the Papillon Client has to be redesigned and reimplemented to make it OS independent.

The client should restart automatically, after the host where it is installed, has rebooted. To make this possible, the client will be installed as a service on the host so that the automatic start is guaranteed.

#### **4.2.3 Power Model Creation Requirements**

Currently, the Papillon power model creation process has to be done following the UCD guidelines, described in the document "Papillon Power Model Generation - PROCESS / METHODOLOGY [Vad14]".

The power model creation process has to be modified because the power model can likewise be applied to calculate the idle power of the server. Due to this, the idle power gathering has to be specified in the power model creation document as well as in the power model generation process itself. This will enable the master to correctly calculate the idle power for the VM. The idle power is one the main information used for calculating the VM power estimation.



## 4.3 Design of the Components

In this chapter, the design of all involved components, which have to be changed to enable the VM support of the Papillon system, are explained and described in detail as follows.

### 4.3.1 Design Papillon Master

The Papillon Master has to be extended to enable the VM support. Because of this, a new power estimation algorithm has to be implemented. Moreover, the database of the Papillon Master database has to be extended as well. Furthermore, the internal GUI has to be extended to ensure that the virtual environment is fully supported.

### 4.3.2 Design Papillon Database

The Papillon Master database will be extended with one new flag. Furthermore, the Papillon Master will have a new algorithm to calculate the consumed power of the VMs. The used algorithm will be selected by using an "if - else" statement which chooses the usual calculation algorithm if the `SERVER` flag is set for the current host otherwise the new algorithm is utilized for calculation.

The power model handling will be extended with a method, which returns the idle power of the current power model, used by the Papillon Master. The database host table is shown in the figure 4.1. The host table of the database will be extended with one completely new field, which describe be the number of VMs on the server.

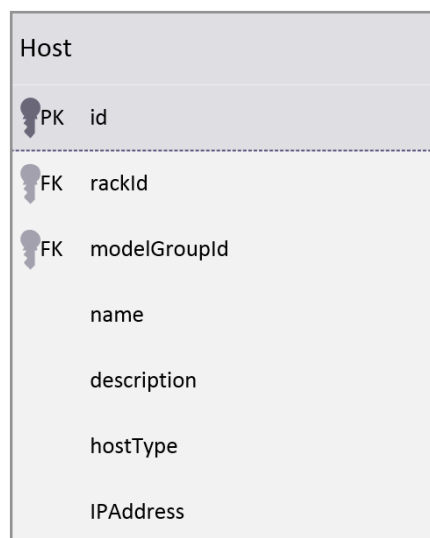


Figure 4.1: The Current Papillon Database Host Table

### 4.3.3 Design Algorithm for VMs

One of the core components of the Papillon system is the algorithm for power estimation. This algorithm has to be extended, to fully support VMs. The new algorithm is implemented on the Papillon Master side. The new algorithm should be applied if the `VIRTUAL` flag is set in the database for the specified host. Otherwise, the algorithm should behave as before.

The design of the new algorithm is explained as follows:

- Read the number of VMs running on the physical server.
- Gather the idle power of the physical server using the power model of the server. The power model has to be created according to the new design.
- Calculate the idle power for the VM, using the number of VMs running on the server.
- Estimate the power consumption of the VM using the power model.
- Subtract the idle power from the physical machine from the estimated value and get the net amount of consumed energy.
- Add the calculated idle power for the VM to the net amount of consumed energy and get the amount of consumed energy for this VM.

### 4.3.4 Design Papillon Client

The Papillon Client, also called agent, is the correspondent part of the Papillon Master. The client is installed and running on the server which should be monitored by the master. The main tasks of the clients are to monitor the system and send packages with system information parameters to the master.

- The client will be written in `Java` to make it OS independent.
- Each OS has different ways to get their system information. Therefore, every OS requires its own implementation for gathering the system parameter. Using the factory design pattern makes it possible, to use one client for different implementations related to the OS.
- The client will have an additional thread which is responsible for sending the packages to the master. The packages consist of the required system information parameters.
- Every sixty seconds, the client will create a new system information package and add this package into the sending queue. Afterwards, the client thread will sleep for sixty seconds.

- The sender thread is autonomously sends the package including the system information parameters. After this is done successfully, the sender thread is blocked by the queue without using any system resources.

### 4.3.5 Papillon Client Start Routine

In this subsection, the start routine from the Papillon Client is explained. A sequence diagram of the client start routine is shown in Figure 4.2. The client is started from the OS. The client start routine begins with the initialization which is explained in detail below.

- Create a Java Logger<sup>1</sup> to write the logging information in a log file. This logger will be used to write the client log messages in the log file. There are different logging level available which control the output written into the file and is related to the defined message log level.
- The given command line arguments are parsed and after success these values are set in the client otherwise an error message is written in the log file and the client exists with an error. The command line arguments are detailed below:
  - Internet Protocol (IP) Address from the Papillon Master
  - Port number from the Papillon Master
  - Host ID which is generated from the Papillon Master and related to this Papillon Client
- The next step is to read the current OS and set it in the client. An enumeration is used to specify the different OSs.

After the client is initialized, the second part of the start routine starts. This part is explained in relation to Figure 4.2 *Sequence Diagram showing the Papillon Client Start Routine* as follows:

1. The SystemInformationFactory is called to create a concrete SystemInformation object. This concrete object is returned and set in the client.
2. A Java BlockingQueue is created and set in the client.
3. A SenderThread is created, initialized and started. The SenderThread and the client share the queue.
4. The final step in the start routine is to start the client thread, which is responsible for the monitoring part. The monitoring part is described in section 4.3.6.

---

<sup>1</sup><http://docs.oracle.com/javase/7/docs/api/java/util/logging/package-summary.html>

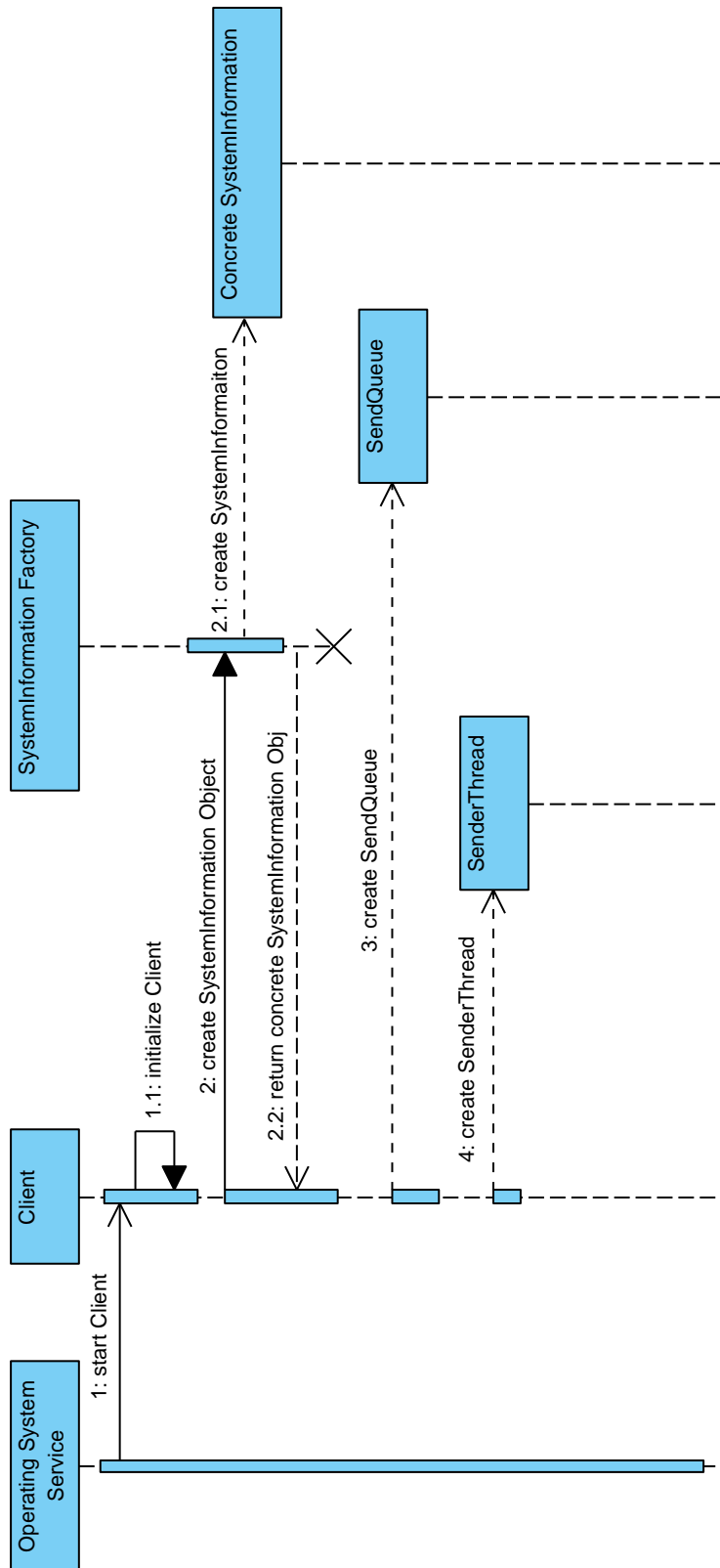


Figure 4.2: Sequence Diagram showing the Papillon Client Start Routine

### 4.3.6 Papillon Client Monitoring and Sending Process

The monitoring process of the Papillon Client is described in this section using the sequence diagram which is shown in Figure 4.3.

This monitoring process is running in an endless loop, all single steps are identified and described as follows:

- The thread sleeps sixty seconds minus the amount of time which was needed for the execution in the previous iteration. The execution time will be calculated from the time stamp at the beginning of the execution and at the end of the execution. This ensures that the thread starts the measuring process after exactly sixty seconds.
- After the thread wakes up, it is going to gather the system information from the OS. Therefore, the concrete SystemInformation object is needed, which includes the OS specific implementation.
- The information received from the OS has to be packed into a StateEntry object.
- This StateEntry object is added to the SenderQueue. The adding part is synchronized which means the operation is thread safe.
- The execution time is calculated and shows the time for the execution described above.
- The thread goes to sleep for sixty seconds minus execution time.

After the monitoring process has added a StateEntry object to the queue, the SenderThread will take it immediately. The SenderThread was waiting in the meantime because of the blocking queue. The access to the queue is synchronized which means that the operations are thread safe. The SenderThread will unpack the data from the StateEntry object and create an XML as well as a JavaScript Object Notation (JSON) object out of it. This object is then sent to the master. The master stores the received information into the database. This information is used to estimate the consumed power of the server.

### 4.3.7 Design Power Model Creation Process

The power model is used for calculating the consumed power of a server. With the VMs, the power model is also used to get the idle power of the server. For this cause, the power model process considers that the power model is now also used to get the idle

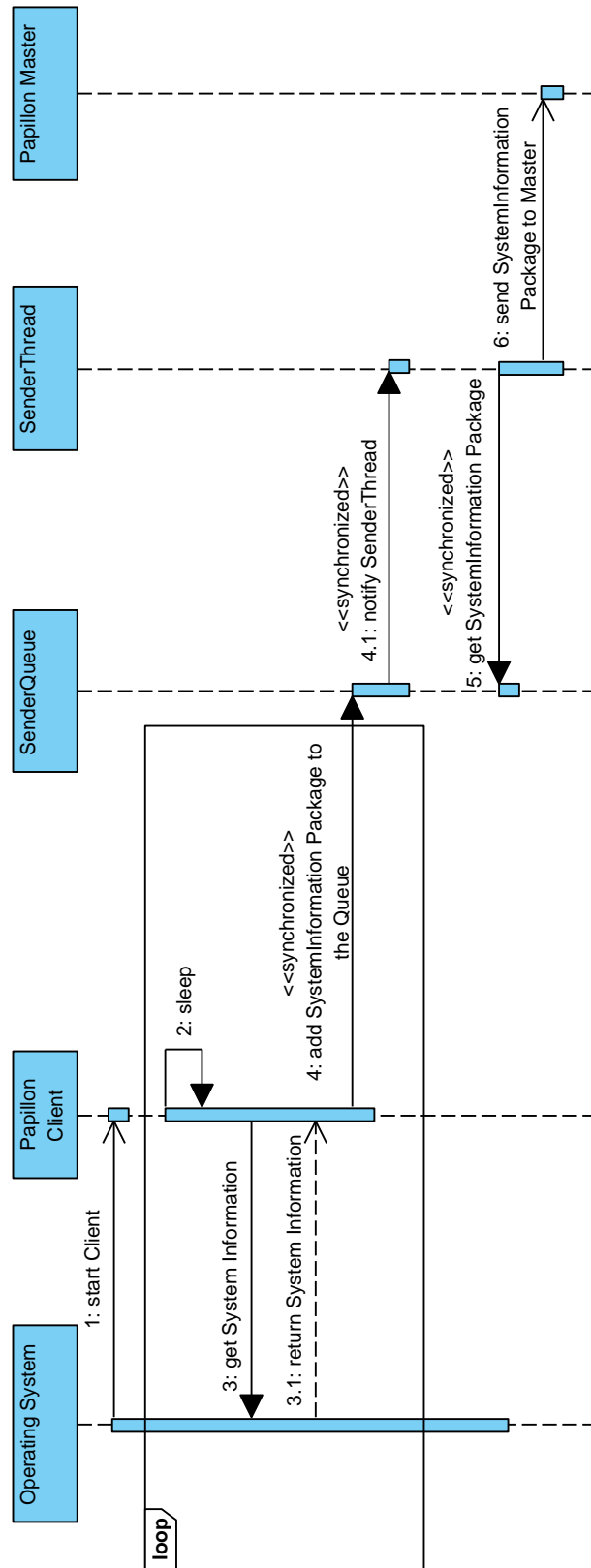


Figure 4.3: Client Monitoring and Sending Process Sequence Diagram

power of the server. The new power model generation process will change in the way that new parts are added. The new parts are added in the beginning of the creation process as well as at the end of the creation process. The part, which will be added, is that the server will be in idle for twenty minutes. This is necessary to ensure that the server idle power is accurate. The server idle power will be calculated by building the sum of the first and last fifteen values of the power model and then calculate the average of these values. The idle power will be saved as member of the Papillon Master.

### 4.3.8 Design Client Installer

The actual client uses an installer script which can be used on Linux based OSs. It is a shell script which uses the Yet Another Java Service Wrapper (YAJSW) library to install the client as a service. This script will be reused and adapted to the new client.

For Windows OSs, a PowerShell script will be created which uses the YAJSW library to install the client as a service on the host. After calling the install script, the installation process of the client starts. The installation of the client as a service is done automatically by using the install script.

## 4.4 Used Tools and Environment

This section explains the used tools and environment that are needed to implement the designed extensions of the Papillon system.

### 4.4.1 Programming Languages

#### Java

Choosing the right programming language was very tough and depended on different things. The focus of this project was to find a programming language which was OS independent so that the client could be run on different OSs. In conclusion, Java<sup>23</sup> was selected to be used for implementing the client, also because of the Papillon Master is implemented in Java.

---

<sup>2</sup>Java is a programming language which is OS independent. Java follows the motto of "Write Once, Run Anywhere (WORA)".

<sup>3</sup><https://www.oracle.com/java/index.html>

## Script Languages

A number of different Script languages are used to fulfill smaller tasks which should be completed quickly. These tasks can include formatting a file, analyzing data, creating and storing files and more. The used languages are described below and their employment is explained at the points where they are employed.

- **Python**<sup>4</sup> is an Object Oriented Programming (OOP) language which includes exceptions, high level dynamic data types, modules and classes. It comes with many interfaces including libraries that run on Uniplexed Information and Computing Service (UNIX) systems and also on Windows systems. The Python Software Foundation License (PSFL)<sup>5</sup> has the Copyright on Python.
- **Ruby**<sup>6</sup> is a script language which is easy to use. Furthermore, Ruby supports the OOP paradigm. Ruby is popular especially because of the web framework Ruby on Rails<sup>7</sup>. Ruby includes exceptions, classes and is very flexible and runs on all UNIX and Windows systems. Furthermore, Ruby is the only programming language where everything is an object which means that everything can have attributes and methods. Ruby underlays the 2-clause Berkeley Software Distribution (BSD)<sup>8</sup>.
- **Batch Script**<sup>9</sup> is a simple language which is used to simplify routines and execute one or more commands. It is written in unformatted text which contains commands. Batch scripts are a proprietary software because it is a part of the Microsoft Windows OS.
- **PowerShell Script**<sup>10</sup> is a task based command-line shell and scripting language built on the .NET Framework<sup>11</sup>. It was developed especially for IT professionals and power user to automate the administration of applications that run on Windows and for the Windows OS.

### 4.4.2 Apache Tomcat

Apache Tomcat<sup>12</sup> is an open source software implementation of the Java Servlet and Java Server Page (JSP) technologies. Apache Tomcat is an open source web-server and web-container developed under the Apache License version 2. Tomcat basically consists of a Servlet Container Catalina, the JSP-Engine and the connector framework Coyote [Fou14].

---

<sup>4</sup><https://www.python.org>

<sup>5</sup><https://www.python.org/psf-landing>

<sup>6</sup><https://www.ruby-lang.org>

<sup>7</sup><http://rubyonrails.org>

<sup>8</sup><http://opensource.org/licenses/bsd-license.php>

<sup>9</sup><https://technet.microsoft.com>

<sup>10</sup><http://microsoft.com/powershell>

<sup>11</sup><https://msdn.microsoft.com/de-de/aa496123>

<sup>12</sup><http://tomcat.apache.org/>



### 4.4.3 System Information

The OS information parameters are the core data to calculate the power consumption. Therefore, gathering this information is required for estimating the power consumption. The first step will be to check what OSs are going to be supported. It was decided by UCD that for now Windows and Linux based OSs are supported.

Linux supports native calls, which gives all needed information. It is easy, simple, and absolutely exact to get the needed system information using `/proc`<sup>1314</sup>.

Windows does not support native calls, therefore, it is necessary to use another method. After some research was done, two libraries were found which are described below.

#### Operating System and Hardware Information (OSHI)

*OSHI is a free JNA-based (native) OS information library for Java. It does not require any additional native DLLs and aims to provide a cross-platform implementation to retrieve system information, such as version, memory, CPU, disk, etc [OSH14].*

OSHI is distributed under the MIT license<sup>1516</sup> and is freely available. This library is easy to use and works as it is predicted. Nevertheless, this library is not useful for the Papillon project, because it is just the basic system information supported. The basic system information is the CPU frequency, the number of cores and processors, the current OS, the whole installed memory and the current memory use. All this information are for the whole system.

The Papillon project requires the exact information from each process. Therefore, this library not useful for this project and therefore not used. Figure 4.4 shows a sample output using the OSHI library.

```
Microsoft Windows 7
2 CPU(s):
Intel(R) Core(TM)2 Duo CPU T7300 @ 2.00GHz
Intel(R) Core(TM)2 Duo CPU T7300 @ 2.00GHz
Memory: 532.1 MB/2.0 GB
```

Figure 4.4: Sample Output using the Oshi Library [OSH14]

#### System Information Gatherer and Reporter (SIGAR)

<sup>13</sup><http://man7.org/linux/man-pages/man5/proc.5.html>

<sup>14</sup>The proc filesystem is a special filesystem, available on UNIX OSs. This pseudo-filesystem can be used as interface for the data structures of the kernel. It can be used to obtain system information from processes in during runtime [mpp].

<sup>15</sup><http://opensource.org/licenses/mit-license.html>

<sup>16</sup>MIT License is an open source license where MIT stands for Massachusetts Institute of Technology.

*One API to access system information regardless of the underlying platform  
[Hyp14]*

The SIGAR application programming interface <sup>17</sup> (API) is a cross platform API which is developed from Hyperic<sup>18</sup>. SIGAR supports most of the common OSs such as the OSs listed below:

- Windows
- Linux
- FreeBSD
- Solaris
- Mac OSX
- HP-UX
- AIX

Moreover, SIGAR support exists across a variety of versions and architectures for each of these OSs. Many different system information parameters are available with SIGAR as well as on the process level. This means that different information is available for each process running on the current OS. Exact monitoring of the OS parameter becomes easy with SIGAR. SIGAR is implemented in pure C with bindings implemented in Java, Perl, and C#. This becomes very important because the Papillon project is implemented in Java. This API is available as enterprise edition but also as Open Source Edition which allows the Papillon project the free use of this tool.

The conclusion of the comparison between OSHI and SIGAR ends with the result that it is better to use the SIGAR library. One possible negative impact using SIGAR library can be that the client size increases due to the additional library.

#### 4.4.4 YAJSW

The YAJSW is a tool to wrap Java applications into services or daemons. It is platform independent and is available under the Lesser General Public License (LGPL)<sup>19</sup> License. YAJSW requires Java version 1.5 or greater [sou14].

The YAJSW is the most powerful Java service wrapper in comparison to Java Service Wrapper (JSW), Apache Commons Daemons (ACD) and Launch4j (L4J). The compar-

---

<sup>17</sup>An application programming interface (API) specifies an interface, which allows interacting with other software component.

<sup>18</sup><https://support.hyperic.com>

<sup>19</sup>The LGPL is a free ware software license published by the Free Software Foundation. This kind of license makes it possible, for companies and developers, to use and integrate the LGPL software parts into their own software.

ison of all wrapper frameworks is shown in table 4.1. Furthermore, the Table 4.1. also shows the reason why the YAJSW was chosen for this work.

<i>Features/Framework</i>	<i>YAJSW</i>	<i>JSW</i>	<i>ACD</i>	<i>L4J</i>
License	LGPL	GPL/Commercial	Apache	BSD/MIT
Run as Windows Service	X	X	X	
Run as UNIX Daemon	X	X	X	
Support Mac OS X	X			
Platform independent Installation	X			
Wrap Java Applications	X	X	X	X
Wrap Groovy Script	X			
Wrap Native Executables	X		X	
Wrap as Executable				X
Portable Configuration	X	X		
Capture and Log Console Output	X	X		
Stop and Reconnect Wrapper	X			
Monitor Application	X	X	X	
Restart Application	X	X	X	
Single Instance Enforcement	X	X		X
Control Process Priority	X	X		
Control Process Affinity	X			
Alert Emails	X	X(Commercial)		
Scripting	X			
Timed Events and Event commands	X	X(Commercial)		
Configuration generator	X			
Define Process Name	X			X
Automatic selection of JVM	X			X
JMX Support	X	X		
System Tray + GUI Console	X			
Windows Cluster Aware	X			
Network Launcher	X			
Support JNLP configuration	X			
Java Webstart Launcher	X			

Table 4.1: Comparison between YAJSW, JSW, ACD and L4J frameworks for wrapping Java Applications [sou14]

#### 4.4.5 Netbeans integrated development environment (IDE)

*NetBeans IDE<sup>20</sup> lets you quickly and easily develop Java desktop, mobile, and web applications, as well as HTML5 applications with HTML, JavaScript,*

<sup>20</sup>integrated development environment (IDE) is a software application which allows software developer to write, compile and debug code. Many IDEs have also lot of other useful functionality which supports the software developer in writing code.

*and CSS. The IDE also provides a great set of tools for PHP and C/C++ developers. It is free and open source and has a large community of users and developers around the world [Cor14b].*

The NetBeans IDE is the official IDE from Oracle<sup>21</sup> Java. The principal characteristics are identified as follows:

- Editor
- Code Analyzer
- Converters to upgrade the Java applications to the new Java language construct
- Debugger
- Find Java Bugs Tool
- Unit Testing
- Profiler to profile projects, files or attached processes to, for example, analyze CPU performance and memory usage

Some advantages of NetBeans are that the software is freely available and also OS independent.

#### 4.4.6 JUnit

One way of testing the source code during the implementation is JUnit<sup>22</sup> testing. JUnit is a framework, which is used in software engineering. It is an instance of Unit test. JUnit is specially developed for Java. The NetBeans IDE has a JUnit integration which makes it easy to create test suites and tests for all classes.

#### 4.4.7 Representational State Transfer (REST)

REST is an abstraction of an architectural principle which allows the design of Web services. The core idea behind this architectural principle is to transfer the data over HTML. This allows communication over a wide range of Web services written in different languages. In other words, Web services that are implemented based on the REST principle are reusable and flexible solutions. REST consists of six architectural constraints which are identified in detail as follows [Fie00]:

- **Client/Server:** this is the most common architectural style for network based applications. It is a separation of concerns from the control logic and the data model.

---

<sup>21</sup><https://www.oracle.com>

<sup>22</sup><http://junit.org>

- **Stateless:** means that the communication between client and server must be stateless. All requests from the client must contain all of the required information so that the server understands the request without using any stored context from the server.
- **Caching:** constraint is that the response data are explicitly labeled as cache-able or non-cache-able to improve network efficiency.
- **Uniform Interface:** is the central feature of its architecture principle which applies the software engineering principle of generality. The four interface constraints defined by REST are as follows:
  - identification of resources
  - manipulation of resources through representation
  - self descriptive messages
  - hypermedia as the engine of application state
- **Layering:** describes an encapsulation of each component which restricts the interacting between components. This constraint is added in order to further improve the behavior of Internet scale requirements.
- **Optional Code-on-demand:** is the last constraint from the REST constraint set and allows the client to download code for execution. This code can be Java Applets, ActiveX controls, scripts and Extensible Style-sheet Language (XSLT)<sup>23</sup>.

---

<sup>23</sup>XSLT is a style sheet language for transforming XML documents in other XML documents, text or HTML documents [Gro14].

# Chapter 5

## Implementation

This chapter describes the implementation of the specified requirements related to Chapter 4 *Design*. The implementation chapter is split into different sections according to the implementation part of the Papillon system. Furthermore, the implementation of additional tools, like for example the installer, is also described in this chapter.

### 5.1 Papillon Master Implementation

The Papillon Master Implementation section is split into different subsections. These subsections describe the different implementation parts of the master. The main changes in the master implementation are to enable the VM support. Based on the Papillon architecture, the design is structured into different components, identified as follows:

- the database
- the internal GUI
- the algorithm for power estimation

All of the different parts of the implementation are described and explained in detail in the following sections.

#### 5.1.1 Papillon Database implementation

To ensure full virtual environment support, the Papillon Database was extended. The field "hostType" became a new entry, which is `VM_SERVER`. This `VM_SERVER` is set if the host is a VM and this will be done when the new virtual host is created. The default value for this field is `SERVER`, which represents a physical server.

Furthermore, if the newly added host was a VM, the number of VMs on the server has to be set. For this reason, a new field in the host table was added, named `nVMs`.

This field represents the number of VMs running on the server. This information is required for the algorithm to calculate the consumed power of VMs. Figure 5.1 shows the new database host table implementation with the valid options for the "hostType" field.

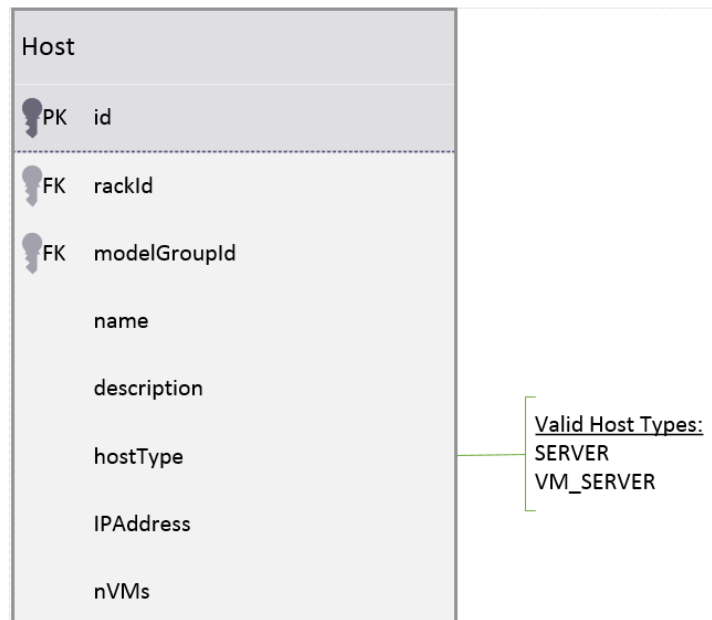


Figure 5.1: The New Papillon Database Host Table Implementation

### 5.1.2 Papillon Master Internal GUI Implementation

The Papillon internal GUI is used to add a new host to the Papillon Master. The new implementation of the internal GUI is shown in Figure 5.2. The changes here are described as follows:

- The new field **Host Type** which reflects the type of the host. The host type can either be `VM_SERVER` or `SERVER`.
- The new field **#VMs** which shows the number of VMs running on the server. This field is only displayed if the server is a virtual server.

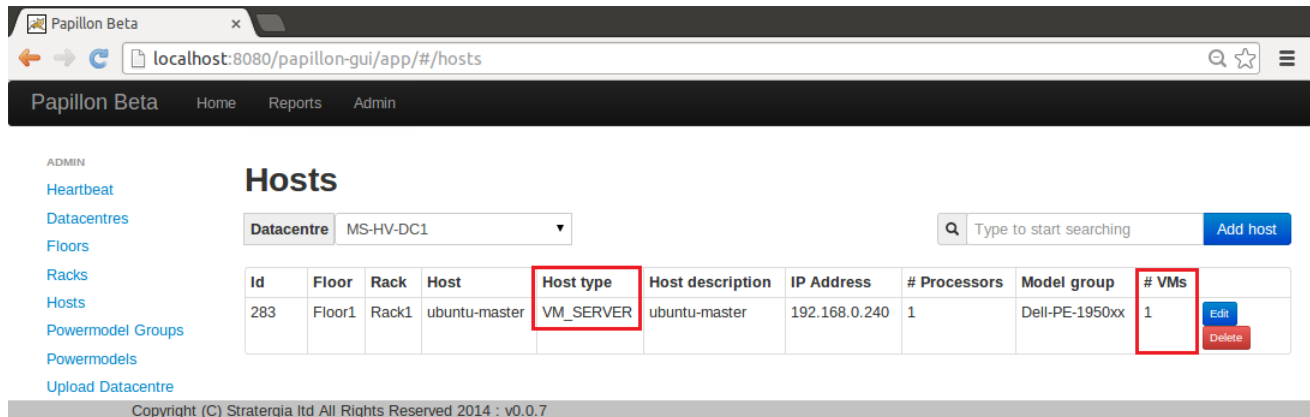


Figure 5.2: Papillon Internal GUI on the Host Tab from a Data Center

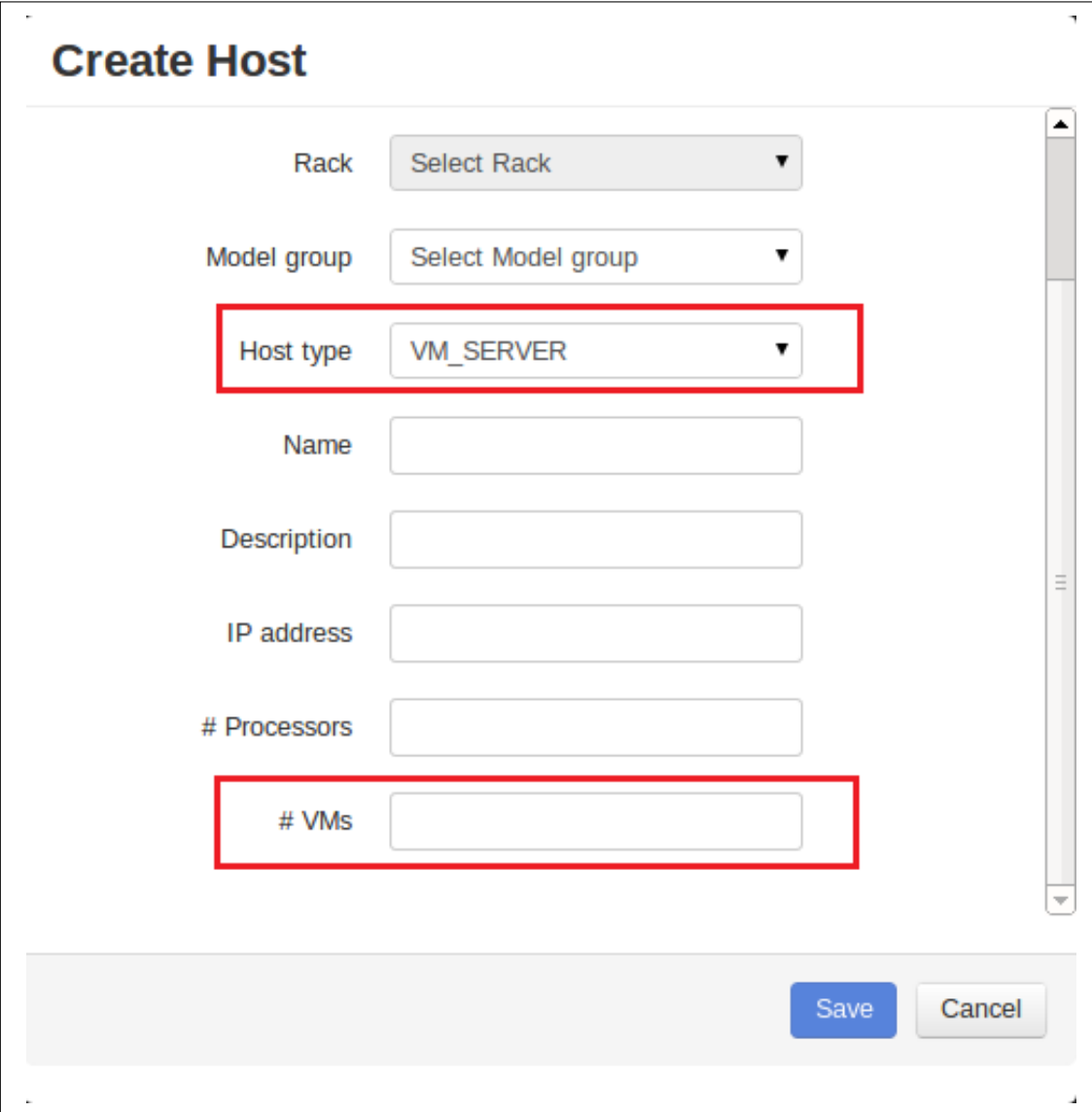
A new host can be added by clicking on the "Add host" button. After clicking on this button, a pop up window is displayed which allows the user to enter the new host information. The "Create Host" pop up window is shown in Figure 5.3 and all of the fields on this window are described in detail as follows:

- **Rack:** is the field where the user or administrator can select the rack to which the newly created host can be added.
- **Model group:** field shows the available power model groups which are available. If the required power model is not available, it has to be added before using the internal GUI. The exact steps on how to do add this will be explained later in this document.
- **Host type:** can be selected using either `VM_SERVER` or `SERVER`.
- **Name:** field must contain the new host name.
- **Description:** field should have a short description of the new host. The user can enter a short description which can consist of information to identify or describe the host.
- **IP Address:** is the IP address of the host which should be created and added.
- **#Processors:** are the number of processors if the server is a physical server otherwise it is the number of virtual processors or cores which is allocated and set by the hypervisor for the VM. This setting is very important because of the power estimation algorithm.



- **#VMs Field:** is only available if the "Host Type" is `VM_SERVER`. The number of VMs running on the server has to be entered in this field. This is a requirement for the power estimation algorithm to work correctly. An incorrect input or no input will make it impossible to estimate the consumed power correctly. If the "Host Type" is `SERVER`, this field is grayed out.

After all fields are completed carefully, the "Save" button can be clicked to store this host configuration otherwise to abort this host specification. After saving, a new host with the entered data is created and added to the selected rack.



The screenshot shows a "Create Host" window with the following fields and controls:

- Rack:** A dropdown menu with "Select Rack" as the current selection.
- Model group:** A dropdown menu with "Select Model group" as the current selection.
- Host type:** A dropdown menu with "VM\_SERVER" selected. This field is highlighted with a red border.
- Name:** A text input field.
- Description:** A text input field.
- IP address:** A text input field.
- # Processors:** A text input field.
- # VMs:** A text input field. This field is highlighted with a red border.

At the bottom right of the window, there are two buttons: "Save" (in blue) and "Cancel" (in gray).

Figure 5.3: Papillon Internal GUI Pop Up Window for Adding a New Host Displaying the Newly Implemented Fields

### 5.1.3 Papillon Data Center Import via XML File

Another method to insert hosts to a Papillon Master is to import a DC using an XML file. This XML file has to have a defined format. An example of the structure is shown in Listing 5.1. The new elements in the XML structure are described as follows:

- **<hostType>** is the new host type which is either `SERVER` or `VM_SERVER`.
- **<VMCount>** which reflects the VMs on the physical server.

Listing 5.1: Example XML File for Importing a Data Center

```

1 <?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
2 <datacenters>
3   <datacenter>
4     <name>test data center</name>
5     <description>description of the datacenter</description>
6     <floors>
7       <floor>
8         <name>test floor center</name>
9         <description>description of the floor</description>
10        <racks>
11          <rack>
12            <name>test rack</name>
13            <description>description of the rack</description>
14            <pdu>12</pdu>
15            <hosts>
16              <host>
17                <name>host number 1</name>
18                <description>description</description>
19                <hostType>VM_SERVER</hostType>
20                <IPAddress>192.168.0.158</IPAddress>
21                <processorCount>1</processorCount>
22                <modelGroupId>186</modelGroupId>
23                <VMCount>10</VMCount>
24              </host>
25              <host>
26                <name>host number 2</name>
27                <description>description</description>
28                <hostType>VM_SERVER</hostType>
29                <IPAddress>192.168.0.158</IPAddress>
30                <processorCount>1</processorCount>
31                <modelGroupId>186</modelGroupId>
32                <VMCount>10</VMCount>
33              </host>
34            </hosts>
35          </rack>
36        </racks>
37      </floor>
38    </floors>
39  </datacenter>
40 </datacenters>

```

This XML file can be imported with the internal GUI using drag and drop. In other

words, the file has to drag-and-drop onto the import field of the internal GUI. This part of the GUI is shown in 6.12.

#### 5.1.4 Implementation of the new Power Estimation Algorithm

The power estimation algorithm is one of the core parts of the implementation according to the design and requirements. The used terms according to the algorithm are detailed as follows:

- *server<sub>total</sub>* is the total consumed power of the physical server.
- The *server<sub>idle</sub>* is the server idle power which is defined as the amount of the consumed power of the server if only the OS is running without any additional operations or workloads. This value is gathered from the power model.
- *vm<sub>power<sub>idle</sub></sub>* is the idle power of the VM. The idle power for a VM depends on the number of VMs running on the physical server.
- *vm<sub>power</sub>* is the amount of consumed power from a VM without idle power.
- The *vm<sub>power<sub>exact</sub></sub>* is the final result of the power consumption of a VM. This VM power exact contains the consumed power from the VM plus the idle power of the VM.
- *#virtual<sub>machines</sub>* is the number of VMs running on the server. This number can be read from the database.

The first step in the new algorithm is to calculate the idle power *vm<sub>power<sub>idle</sub></sub>* of the VM. This value depends on the number of VMs *#virtual<sub>machines</sub>* running on the physical server. The number of VMs running can be queried from the database and the server idle power *server<sub>idle</sub>* is calculated from the power model. The VM idle power *vm<sub>power<sub>idle</sub></sub>* calculation is shown as follows:

$$vm\_power_{idle} = \frac{server_{idle}}{\#virtual\_machines} \quad (5.1)$$

To get the VM power *vm<sub>power</sub>* from the current VM without the server idle power *server<sub>idle</sub>*, the server idle power *server<sub>idle</sub>* will be subtracted from the total consumed power *server<sub>total</sub>* of the server. The exact calculation is shown as follows:

$$vm\_power = server_{total} - server_{idle} \quad (5.2)$$

The final step is to calculate the exact consumed power  $vm\_power_{exact}$  of the VM. This is completed by summing the consumed power  $vm\_power$  from the VM with the idle power  $p_{idle}$  from the VM. The calculation is shown as follows:

$$vm\_power_{exact} = vm\_power + vm\_power_{idle} \tag{5.3}$$

The  $vm\_power_{exact}$  represents the exact amount of consumed power of the VM. This resulting value is stored in the Papillon database and represents the consumed power of the current VM. For a better understanding of the algorithm a work flow diagram is shown in Figure 5.4.

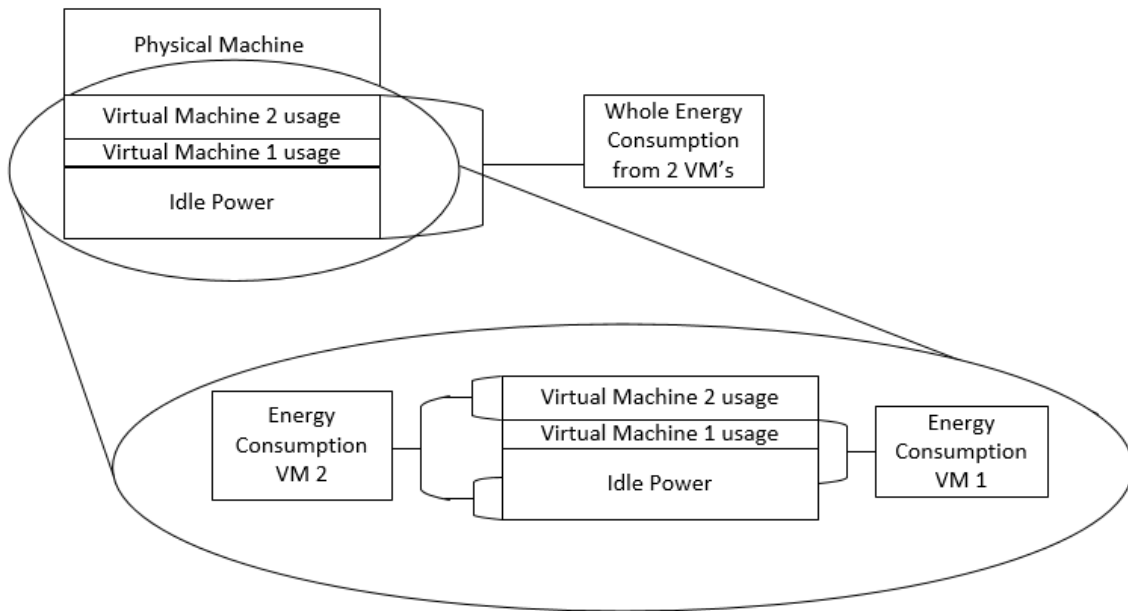


Figure 5.4: Design of the new Algorithm and its Work Flow

The  $vm\_power_{exact}$  value is shown in the internal Papillon GUI as a power profile. The GUI is used to query these values from the database. The consumed power of a VM is calculated and stored afterwards. After clicking on a sample point, a pie chart is shown which details what this consumed power value consists of, meaning the three most active applications on this VM if the current host is a VM. Otherwise the pie chart shows the server's power consumption and the pie chart shows the top three applications of the server. An example of the internal GUI with sampled data and the pie chart is shown in Figure 5.5.

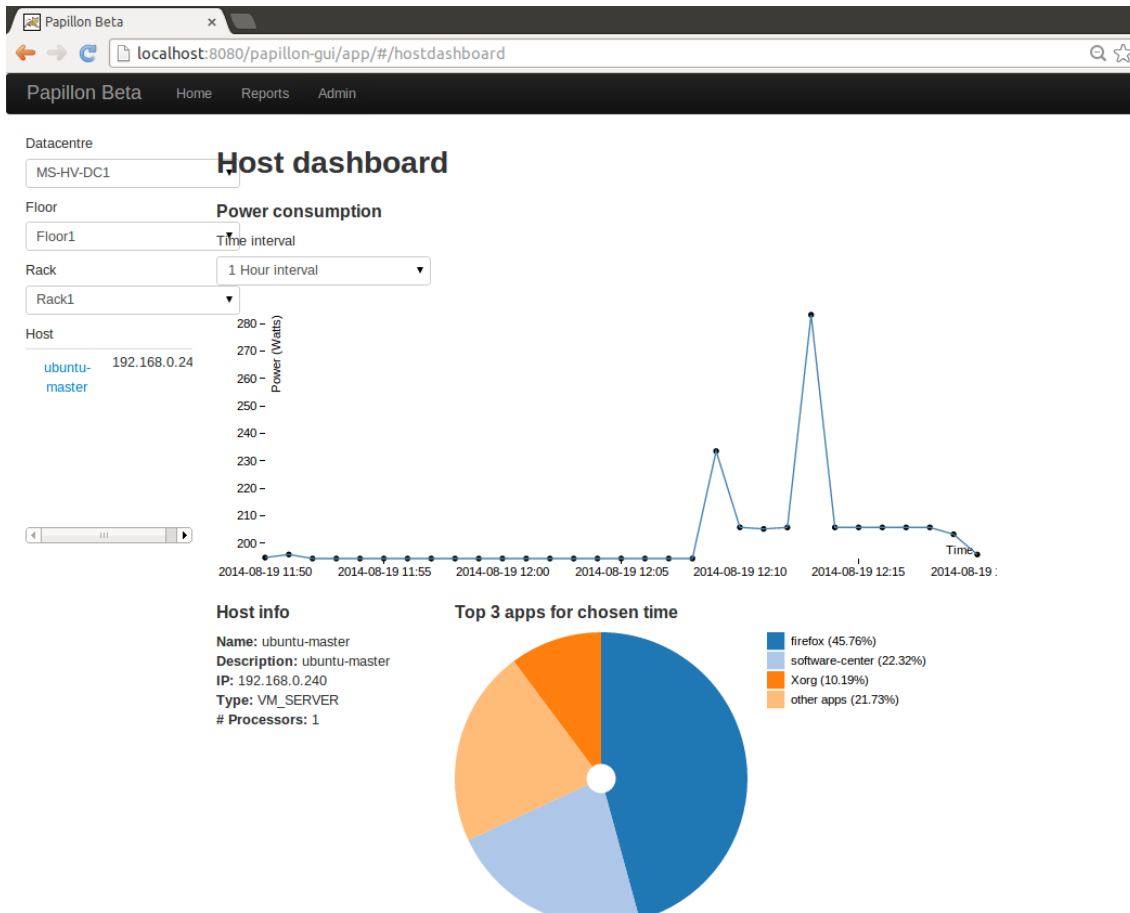


Figure 5.5: Papillon Internal GUI showing Pie Chart for a Sample Point

## 5.2 Papillon Client Implementation

This section describes the implementation of the Papillon Client in detail. A class diagram is used to explain the implementation as well as two sequence diagrams which should explain how the interaction between classes operate together with the workflow. The first part is an overview and explanation of the important parts of the client implementation.

### 5.2.1 Papillon Client Implementation Overview

The Papillon Client implementation is written in Java to allow for OS independence and implemented using the NetBeans IDE. The main parts of the Papillon implementation are shown in a class diagram in Figure 5.6 and explained in detail as follows:

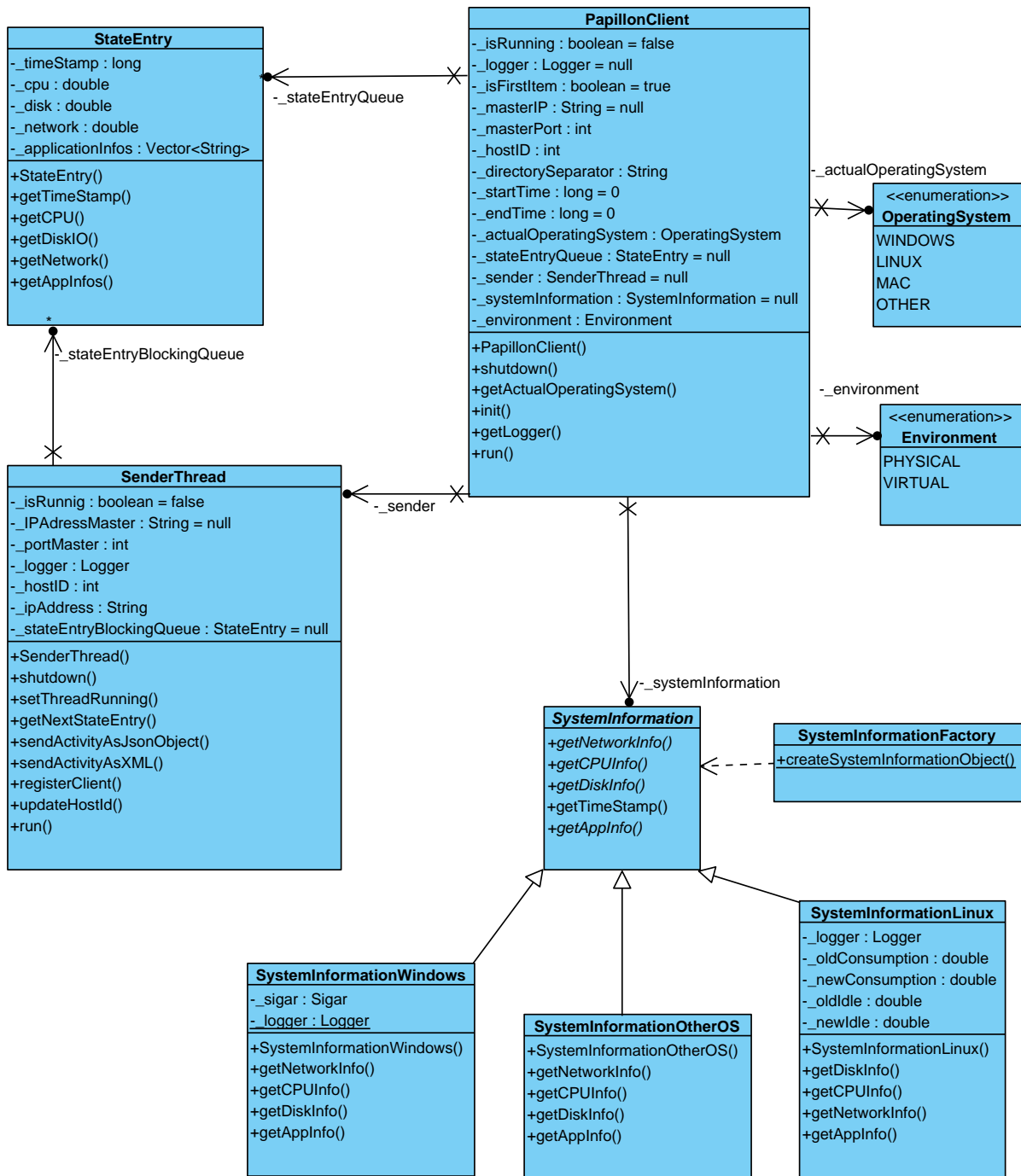


Figure 5.6: Papillon Client Class Diagram with the Main Parts of the Client Implementation

- **The Papillon Client** is the core of the system. It is a very complex part which is explained in detail in Section 4.3.5. This class is responsible for gathering the system information parameters, packing this information into a StateEntry package and adding it to the sender queue. Furthermore, the client ensures that

after 60 seconds the monitoring process starts again.

- **SystemInformation** is an abstract class that is the base class of the concrete SystemInformation objects. A factory design pattern is used to create the concrete SystemInformation. It is a part of the creational pattern and one advantage is decoupling the caller from the implementation of the concrete classes. Furthermore, the system becomes very easy to extend if another OS support is required.
- **Concrete SystemInformation** is the derived class from the SystemInformation class which is created by the SystemInformationFactory class. This concrete SystemInformation object includes the implementation to gather the system information from the current OS.
- **SystemInformationFactory** is the factory class which is called from the client and returns the concrete SystemInformation object.
- **StateEntry** class is a package of information which is created in the client using the SystemInformationObject related to the OS. A StateEntry object consists of different parameters which are described as follows:
  - The time stamp from the OS in milliseconds
  - The CPU consumption
  - Disk and network I/O
  - Some additional information which is required for the client
- **BlockingQueue**<sup>1</sup> is a Java interface that has some special features which became handy in this case. For example, the thread will be blocked from the queue when it tries to take an element from the empty queue. The thread will be blocked until a new element is added to the queue, null elements are not possible.
- **SenderThread** is a self contained thread which is started by the client. Both the SenderThread and the client are sharing one BlockingQueue with StateEntry. The SenderThread is waiting for a new element to be added to the queue. In the meantime, it is sleeping and not consuming any system resources. When the client adds a StateEntry object, the SenderThread will take it, make a few checks and format the given data into XML format as well as into a JSON object.

Next, the data packet is sent to the Papillon Master. All of these operations happen independently from the client. After the information package is sent, the

---

<sup>1</sup><http://docs.oracle.com/javase/7/docs/api/java/util/concurrent/BlockingQueue.html>

SenderThread will try to take the next StateEntry object from the queue. If there is none it has to wait.

### 5.3 Changes in the Papillon Power Model Generation

The power model is one of the core parts of the Papillon system. To extend the system successfully, the power model generation process had to be extended. The main change which had to be done in the creation part was to ensure that it is possible to obtain an accurate system idle power from the power model.

The idle power of the server is a very important part of the power estimation for VMs. This was the main reason why we had to make sure that we had an accurate system idle power from the generated power model. The power model generation process is done with a power model generation tool which is developed by UCD. For each physical server with different hardware, a new power model has to be created. The process of how a power model is created is described in detail below.

- At the beginning of the power model generation process, the power model generation tool will wait twenty minutes before the benchmarks start. During this time the server runs in idle and the first 20 values are stored as the idle values. The amount of twenty minutes was chosen because every server is different in relation to how they behave during operation in real idle mode. The time is different, depending on the hardware manufacturer and also depending on the OS running on the server. These twenty minutes ensure that the server is really running in idle mode so that the accuracy of the idle value is correct.
- The next step is to run different benchmarks and record the consumed power with the system information. These recorded values are the used CPU consumption, the disk I/O, the network I/O and the memory I/O. This process takes approximately six hours.
- After successfully finishing the benchmark part, the power model generation tool waits another twenty minutes to allow the server to come down to the idle mode. This is the second time where the server idle power is recorded.
- The final step of the power model generation tool is to store the recorded values in a power model file which is an XML file.

This power model file can be imported into the Papillon database and used for power estimation for physical server as well as for VMs.



## 5.4 Papillon Client Installer for Windows Implementation

The installation of the Papillon Client for both the physical or virtual server should be as easy as possible. For Linux based OSs, there was already a shell script developed to install the client as service on the system. The advantage of installing the client as a service on the system is that after a reboot the client starts by itself and can start monitoring the system and sending the information packages to the Papillon Master.

The installer for Windows systems consists of three different scripts to install the Papillon Client as a service. The installing process is using the YAJSW library for installing the client properly. The folder named "ClientInstaller" contains the required files for installation and the YAJSW library. Furthermore, the client installer package contains the folder structure which is necessary for successfully installing the Papillon Client. The installation process is described in detail as follows:

- First step is to open a Windows command window as administrator. There has to be administrator privileges available otherwise the installation process will fail.
- Navigate to the directory named "ClientInstaller" and enter the folder named "client".
- This folder contains a file setup.bat. Open this file with arguments described as follows:
  1. *-ip* < **IPAddress** > where < **IPAddress** > is the IP address of the Papillon Master.
  2. *-po* < **Port** > where < **Port** > is the port of the Papillon Master.
  3. *-id* < **HostID** > where < **HostID** > is the host identification number of the host. This number can be found on the Papillon Master because before installing the client, the client has to be included on the Papillon Master. After successfully adding a new host to the master, the host ID is created.
- If the batch file is executed with incorrect argument usage an error message is printed which displays the correct usage.
- An example call of the batch file can be:  
*setup.bat -ip 192.0.0.1 -po 8080 -id 211*

- At the end of the installation process, a message is printed in the command window, which shows if the installation was successful or not. If the installation was successful, the service is started immediately. If the installation is unsuccessful, the service will not be started at all.
- If an error occurs, the service has to be uninstalled using the `uninstall.bat` file in the YAJSW folder. This is necessary in order to remove the partial installation from the system and to make sure the next installation process can start from scratch.

The installer package consists all required files including the YAJSW library. The script files for installing the Papillon Client are described below.

- **setup.bat:**

This is the main file which is used for installing. This file has to be called with the arguments explained above. If the script is executed with an incorrect argument usage, an error message is displayed in the windows terminal otherwise the PowerShell script will be executed. By default, the script execution is disabled on Windows and for this reason this file has to be executed with administrator rights because the script execution will be enabled. The source code is shown in Appendix B Listing 2.

- **installPapillonClient.ps1:**

This PowerShell file is needed to install the Papillon Client and called from the `setup.bat`. The first thing that happens here is that the script will start a process using the helper file. This is necessary to get Process Identification Number (PID) from the newly started process which is the Papillon Client. This PID is required from the YAJSW library to install the client as a service properly.

After successfully installing, the service is started and the client starts to monitor the system and send information packages to the Papillon Master server. If an error occurs, a message is displayed in the Windows command window. The source code is shown in Appendix B Listing 3.

- **getCorrectPid.bat:**

This batch file is a helper script which starts a Papillon Client process called from the PowerShell file named `installPapillonClient.ps1`. This is needed to get the PID by the client which is used from the YAJSW library for the installation process. The source code of the helper script is shown in Appendix B Listing 1.

## Chapter 6

# Tests and Experimental Results

This chapter is all about tests, verification, evaluation and the results. These tests were made during different phases, which start with the research phase, followed by the implementation phase and lastly, the verification phase. An overview, including a short description of each segment is presented in Section 6.1 *Overview*.

### 6.1 Overview

The development process is split into three main phases. The tests related to the phases are identified and explained in this chapter. A short overview is described below:

**Research Phase:** Section 6.2 *Verification of the CPU related Consumption from Virtual Machines and their Hosts* is part of the research and design phase in the development process in this thesis. This phase has to do with checking the interaction and behavior between the host and VM CPU usage and consumption. The results from this phase will be used to develop an accurate solution for the power analysis of VMs and the associated processes running on them. These tests will verify different OSs together with different virtualization software tools.

**Implementation Phase:** During the Papillon Client implementation, as well as at the end of the implementation, JUnit tests are used to make sure that the written code is working properly. These tests are explained in Section 6.3 *JUnit Tests*.

**Verification Phase:** This final phase consists of a number of different sections which are identified and described as follows:

- It is necessary for the new client implementation to be very light with respect to the resources utilized on the host server such as memory footprint and the amount of CPU resource required. For this reason, the resources on the

host server are monitored and analyzed by using the NetBeans Profiler on different OSs. All tests and results are described in detail in Section 6.4 *Performance Impact Evaluation of the Papillon Client on different Operating Systems*. These tests are part of the implementation as well as the verification phase in the development process.

- In Section 6.5 *Verifying the Accuracy of the Estimated Power Consumption of the Papillon System on a Virtual Environment* the most relevant test according to this work is described and explained in detail. This test is about identifying accuracy of the calculated power consumption using the virtual environment for the Papillon Master and Client. This test is part of the verification phase in the development process and all results are described in detail. Moreover, the tool chain and setup used for these tests are also explained in detail.
- A stress test is employed to the Papillon Master which is explained in detail in Section 6.6 *Papillon Master Stress Test*. Stress test means in this case to identify the number of clients that are communicating in a specified time range to the master. In other words, the number of clients the master can handle in a specified time range.
- Papillon Master allows a DC to be imported using an XML file. This XML file includes all information required such as floors, racks and hosts. To verify the number of possible clients that can be imported with one XML file into the master, the Section 6.7 *Import Data Center via XML File* is used to describe this in detail and to analyze the results.
- Section 6.8 *Tool to identify the Number of Possible Virtual Machines for Given Server Setup* describes a tool which is developed to identify the maximum number of VMs running on a server with respect to a specified hardware setup of the server.

## 6.2 Verification of the CPU related Consumption from Virtual Machines and their Hosts

One of the key requirements during the research phase was to determine the CPU usage of a VM. In other words, how the CPU of a VM interacts with the host machine's CPU. For this experiment, different OSs are used together with several virtualization tools. The tests were done with a small test application, which is used to increase the CPU utilization in the VM, which also increases the CPU utilization of the host machine. Another important parameter for this experiment was to check the power consumption using different hypervisors.

### 6.2.1 Hardware and Software Setup

To verify the interaction between hypervisor and guest, several different Operating Systems are used. These Operating Systems are Windows 7, Windows 8 and Linux Ubuntu 13.10. A number of different hypervisors are used for this evaluation. The different hypervisors used are listed below:

- VMware Player
- VMware Workstation
- Oracle VirtualBox

It is possible to change the settings of the hypervisor such as the number of cores or memory allocated to the VM. The following tests were performed with different settings. Additionally, a small script was written to simulate activity on the guest. The CPU usage of the host and the guest was monitored by different tools depending on the OS. On Windows OSs, the Windows system performance meter was used while the gnome system monitor<sup>1</sup> was used for Ubuntu systems.

The hardware platform for this test is detailed as follows:

- **Laptop Lenovo T430s**<sup>2</sup>
  - **Operating System:** Microsoft Windows 8.1 Professional
  - **Processor:** Intel Core i5-3320M CPU 2.60GHz
  - **Random Access Memory (RAM):** 8 GB
  - **OS Architecture:** 64-bit

### 6.2.2 Test Scenario

The hardware described above is used to run different hypervisors. Each hypervisor will have two VMs installed with the mentioned OSs. Only one VM is running on each test. A script is executed in the running VM which stresses the CPU on three different stages. The CPU utilization is monitored on both the VM and the host.

All of the results are shown in Table 6.1, Table 6.2, and Table 6.3.

---

<sup>1</sup><https://apps.ubuntu.com/cat/applications/quantal/gnome-system-monitor/>

<sup>2</sup><http://shop.lenovo.com/us/en/laptops/thinkpad/t-series/t430s>

<i>Hypervisor</i>	<i>Operating System</i>	<i>Cores</i>	<i>CPU Usage Guest</i>	<i>CPU Usage Host</i>
VMware Player	Windows 7	1	idle	0%
VMware Player	Windows 7	1	50%	12,5%
VMware Player	Windows 7	1	100%	25%
VMware Player	Windows 7	2	idle	0%
VMware Player	Windows 7	2	50%	25%
VMware Player	Windows 7	2	100%	50%
VMware Player	Windows 7	4	idle	0%
VMware Player	Windows 7	4	50%	50%
VMware Player	Windows 7	4	100%	~ 98%
VMware Player	Ubuntu 13.10	1	idle	0%
VMware Player	Ubuntu 13.10	1	50%	12,5%
VMware Player	Ubuntu 13.10	1	100%	25%
VMware Player	Ubuntu 13.10	2	idle	0%
VMware Player	Ubuntu 13.10	2	50%	25%
VMware Player	Ubuntu 13.10	2	100%	50%
VMware Player	Ubuntu 13.10	4	idle	0%
VMware Player	Ubuntu 13.10	4	50%	50%
VMware Player	Ubuntu 13.10	4	100%	~ 97%

Table 6.1: Table with Results using VMware Player

<i>Hypervisor</i>	<i>Operating System</i>	<i>Cores</i>	<i>CPU Usage Guest</i>	<i>CPU Usage Host</i>
VMware Workstation	Windows 7	1	idle	0%
VMware Workstation	Windows 7	1	50%	12,5%
VMware Workstation	Windows 7	1	100%	25%
VMware Workstation	Windows 7	2	idle	0%
VMware Workstation	Windows 7	2	50%	25%
VMware Workstation	Windows 7	2	100%	50%
VMware Workstation	Windows 7	4	idle	0%
VMware Workstation	Windows 7	4	50%	50%
VMware Workstation	Windows 7	4	100%	~ 98%
VMware Workstation	Ubuntu 13.10	1	idle	0%
VMware Workstation	Ubuntu 13.10	1	50%	12,5%
VMware Workstation	Ubuntu 13.10	1	100%	25%
VMware Workstation	Ubuntu 13.10	2	idle	0%
VMware Workstation	Ubuntu 13.10	2	50%	25%
VMware Workstation	Ubuntu 13.10	2	100%	50%
VMware Workstation	Ubuntu 13.10	4	idle	0%
VMware Workstation	Ubuntu 13.10	4	50%	50%
VMware Workstation	Ubuntu 13.10	4	100%	~ 97%

Table 6.2: Table with Results using VMware Workstation

<i>Hypervisor</i>	<i>Operating System</i>	<i>Cores</i>	<i>CPU Usage Guest</i>	<i>CPU Usage Host</i>
Oracle VirtualBox	Windows 7	1	idle	0%
Oracle VirtualBox	Windows 7	1	50%	12,5%
Oracle VirtualBox	Windows 7	1	100%	25%
Oracle VirtualBox	Windows 7	2	idle	0%
Oracle VirtualBox	Windows 7	2	50%	25%
Oracle VirtualBox	Windows 7	2	100%	50%
Oracle VirtualBox	Windows 7	4	idle	0%
Oracle VirtualBox	Windows 7	4	50%	50%
Oracle VirtualBox	Windows 7	4	100%	~ 96%
Oracle VirtualBox	Ubuntu 13.10	1	idle	0%
Oracle VirtualBox	Ubuntu 13.10	1	50%	12,5%
Oracle VirtualBox	Ubuntu 13.10	1	100%	25%
Oracle VirtualBox	Ubuntu 13.10	2	idle	0%
Oracle VirtualBox	Ubuntu 13.10	2	50%	25%
Oracle VirtualBox	Ubuntu 13.10	2	100%	50%
Oracle VirtualBox	Ubuntu 13.10	4	idle	0%
Oracle VirtualBox	Ubuntu 13.10	4	50%	50%
Oracle VirtualBox	Ubuntu 13.10	4	100%	~ 94%

Table 6.3: Table with Results using Oracle VirtualBox

### 6.3 JUnit Tests

During the client implementation phase JUnit tests were used to verify the implemented Java code. NetBeans includes JUnit support which allows test suites and test cases to be generated for the classes. The generated test suites were also extended by manually written tests to complete the code coverage. The library applied for these JUnit tests was **JUnit 4.10**<sup>3</sup>.

The Papillon Master already has its own test suites which were extended to cover the newly implemented functionality.

On top of this, the client code itself is using asserts as pre- and post conditions used in important methods. These asserts are disabled by default from the Java compiler. Compiling the project with "-ea" compile flag enables the assert, which was used during the implementation. With this testing technique it is much easier and faster to find implementation failures.

After successfully implementing and testing the Papillon Client, the release version was compiled with disabled asserts.

---

<sup>3</sup><http://junit.org>

## 6.4 Performance Impact Evaluation of the Papillon Client on different Operating Systems

### 6.4.1 Overview

To ensure that the Papillon Client does not use too many system resources, the monitoring and profiling of the running client has to be verified. The Papillon Client should be a lightweight tool which requires small impact on the performance on the actual OS. The reason for keeping the client simple and light is that it should have a negligible affect on the running system.

### 6.4.2 Profiling the Papillon Client using NetBeans Profiler

These tests are completed with the NetBeans Profiler which can profile and analyze system parameters like the memory (heap) which is used in the profiled application. The heap is shown as maximum heap and used heap. Another memory diagram shows the relative time which was spent in garbage collection during the profiling session. Furthermore, the CPU consumption of the application is also profiled and analyzed. The number of threads and classes loaded, and their activity is also included in the profiling session.

The profiling was done on different OSs to see the difference in the results and the influence of the OS by the client. Figures A.1, A.2, A.3 and A.4 show the VM Telemetry Overview provided by the NetBeans Profiler. A more detailed explanation and description about the figures is available in Section A.1 *Description*. Table 6.4 shows the results given by the NetBeans Profiler for different OSs.

### 6.4.3 Results and Evaluation

<i>Operating System</i>	<i>physical/virtual</i>	<i>cores</i>	<i>RAM</i>	<i>av. CPU</i>	<i>av. RAM</i>
Windows 8.1 64 bit	physical	4	8	<1%	12.7 Mb
Ubuntu 14.04 64 bit	physical	4	8	<1%	6.8 Mb
Windows 7 64 bit	virtual	2	2	<1%	8.4 Mb
Mint 17 Linux 64 bit	virtual	2	2	<1%	7.2 Mb

Table 6.4: Table with Client CPU and RAM Usage on different Operating Systems, Results from NetBeans Profiler

The analysis of the results shows that the client correlates to the requirements of using a very small amount of system resource usage like memory and CPU. The client will have almost no influence on a system during operation.



## 6.5 Verifying the Accuracy of the Estimated Power Consumption of the Papillon System on a Virtual Environment

In this section, the process to verify the accuracy of the Papillon system on a virtual environment is described, explained, tested and analyzed. Subsection 6.5.1 *Overview of the Accuracy Verifying Process*, gives an overview of the used methodology.

### 6.5.1 Overview of the Accuracy Verifying Process

An example test setup is shown in figure 6.1. Related to this figure, the setup consists of different parts, hardware and software, which are necessary to evaluate the accuracy. A review of all the components in respect to Example Test Setup with three Virtual Machines, a Power Meter and a Test Computer are described as follows:

1. A server is the physical hardware where the virtual environment is installed.
2. A native hypervisor is installed onto the server.
3. The hypervisor is the software layer between server hardware and VMs.
4. In this example there are three VMs running with different OSs where one of the VMs is the Papillon Master.
5. All VMs have clients installed, which monitors the system, and report the data to the Papillon Master. The master also has a client installed, which monitor and reports to itself.
6. The master stores all information received from the clients and estimates the consumed power of the server.
7. A power meter is plugged in between the power source and server. We can now measure the exact power consumption of the server.
8. A script is executed on the test computer which collects the power data from the VMs and the power meter.
9. On the test computer a script is executed which collects the power data from the VMs and the power meter.
10. In other words, the consumed power amount of all three VMs together should be equal to the real measured amount read from the power meter if the measure time was the same.

With this information it is possible to calculate the accuracy of the estimated values. The accuracy of this system is measured with the average error and the standard deviation.

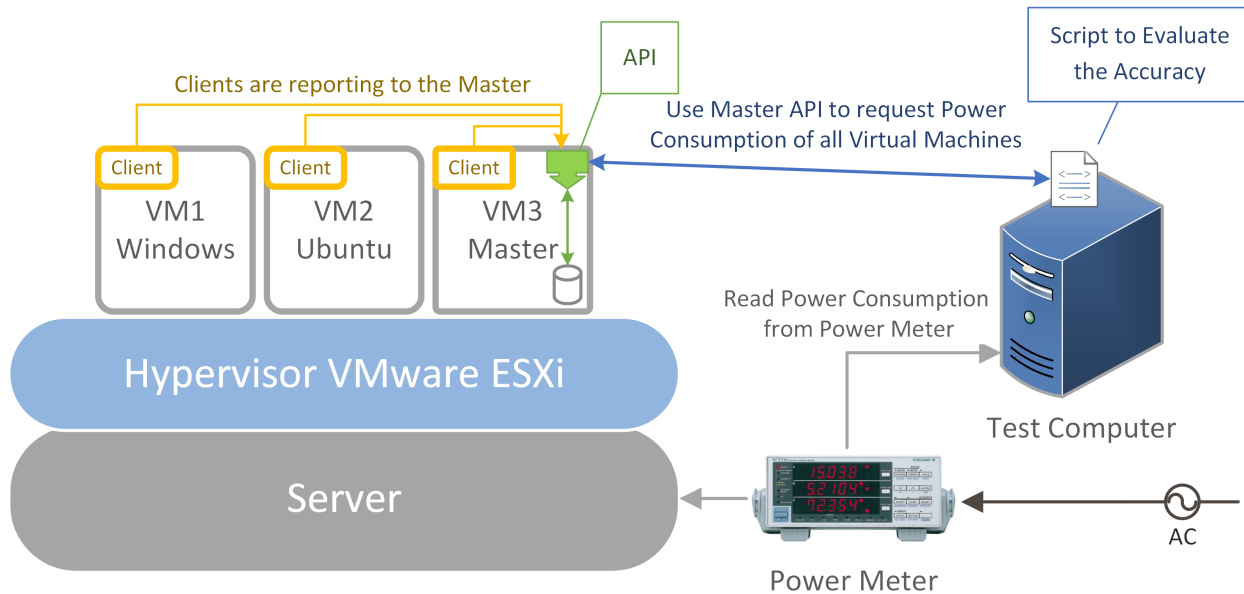


Figure 6.1: Example Test Setup with three Virtual Machines, a Power Meter and a Test Computer

### 6.5.2 Used Hardware

- **Server: Dell PowerEdge 1950** This server is used to run the VMs which means it represents our virtual environment.
- **Power Meter: Yokogawa WT210** The power meter is used to measure the energy consumption from the server. Figure 6.2 shows the Yokogawa WT210 power meter which is used to measure the amount of consumed power of the server.



Figure 6.2: Yokogawa WT210 Power Meter [COR14c]

### 6.5.3 Used Software

- **VMware ESXi:** Is a virtualization tool for server offered by VMware. This is described in detail in Section 2.3.1 *VMware Virtualization*.
- **VMware VSphere:** Is a remote control and management tool for VMware ESXi. This tool is described in detail in Section 2.3.1 *VMware Virtualization*.
- **Python:** A script used for comparison of values is implemented in Python. This script gets the actual power values from the power meter and compares it with the calculated values from the database. The results are saved in a text file. This script uses a very small time window which is plus minus five seconds for four VMs. The first thing what the script does is to synchronize the start / measure time with the entry in the database. Then the five-second time window is used to getting the data from the database. This means the script generate some SQL queries with the actual time stamp plus-minus five seconds to get the result. If there are only two machines, the time gap is approximately two seconds but with increasing the number of VMs also the time gap increases. Four VMs are using a time gap smaller than five seconds.
- **Ruby:** A script is implemented ruby, which is used to format the results file for importing it from Matlab.
- **Matlab:** This tool is used to visualize the results data from the value comparison of the measured and calculated values.

### 6.5.4 Problems

One of the biggest problems in this test was the time gap from the reporting clients. This occurs because every single client needs a different amount of time to start. This problem is illustrated in figure 6.3. Because of the different measuring time of each VM and the power meter as well, the values, which are used for the calculation, vary. The measured values vary because the server does not have a constant energy consumption.

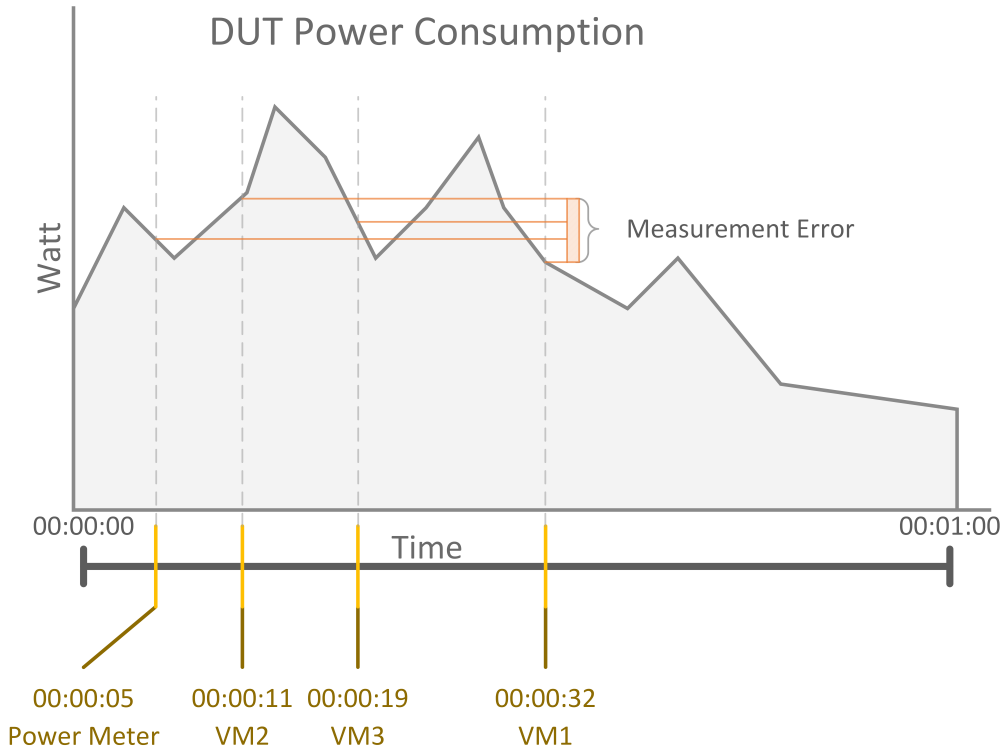


Figure 6.3: Measurement Error if Clients and Power Meter not Time Synchronous

In the first tests, the time gap was almost sixty seconds, which make the result very inaccurate. The solution, to reduce the time gap to increase the accuracy is to extend the client. The client was extended, so that after it starts, it will wait until the next full minute and starts, and then begin with the monitoring and sending the package process. To increase the accuracy even more, also the execution time is recorded and subtracted from the sixty second sleep time. The new sleep time will be calculated as shown in the equation 6.1. The start and end time from the program execution will be measured and then from the initial sleep time subtracted so that all together the reported time occurs at sixty seconds intervals. In the equation, used parameters are milliseconds because it is more accurate to stop the time in milliseconds.

$$sleeptime = 60000 - (stoptime - starttime) \quad (6.1)$$

### 6.5.5 Test Scenarios

To increase the accuracy of these results, compared to a real DC usage, different test scenarios are considered. Different test scenarios in the sense of different number of VMs, OSs, different amount of resources such as cores added to the VM and the different

workloads of each VM. Table 6.5 shows the different test scenarios using a matrix that includes the number of VMs, the specific OS and the amount of cores used.

<i>Scenario</i>	<i>No. VMs</i>	<i>Operating Systems</i>	<i>No. Cores</i>
Scenario I	4	Ubuntu 12.04 64 bit LTS	1
		Mint Linux 64 bit	1
		Windows 8.1 64 bit	1
		Windows 7 64 bit	1
Scenario II	4	Ubuntu 12.04 64 bit LTS	2
		Mint Linux 64 bit	2
		Windows 8.1 64 bit	2
		Windows 7 64 bit	2
Scenario III	4	Ubuntu 12.04 64 bit LTS	4
		Mint Linux 64 bit	4
		Windows 8.1 64 bit	4
		Windows 7 64 bit	4
Scenario IV	4	Ubuntu 12.04 64 bit LTS	4
		Mint Linux 64 bit	2
		Windows 8.1 64 bit	8
		Windows 7 64 bit	1
Scenario V	2	Ubuntu 12.04 64 bit LTS	4
		Windows 7 64 bit	1
Scenario VI	2	Ubuntu 12.04 64 bit LTS	4
		Mint Linux 64 bit	2
Scenario VII	1	Ubuntu 12.04 64 bit LTS	4
Scenario VIII	1	Ubuntu 12.04 64 bit LTS	1

Table 6.5: Table with used Scenarios together with the Scenario Configuration

The benchmarks that are used for the different situations depend on the OS. On Linux OSs the tiobench<sup>4</sup> is used to stress the Linux systems. For Windows, PCMark<sup>5</sup> is used to have random activity on the system which makes the results more accurate. All of these benchmarks are simulating activity on the VM which should be similar to the real usage of a VM in a DC. This is also called "A Real World Scenario". The Papillon Master and the Papillon Client run on the Ubuntu 12.04 64 bit LTS. The Papillon Client reports the consumed data to itself.

The python script synchronizes the start time with the database entries from the clients to make the results more accurate. After the time synchronization, the script gathers the time matching data from the database which is coming from the different clients on

<sup>4</sup>The tiotest is a file system test for Linux developed by Mika Kuoppala especially designed to test I/O performance with multiple running threads [KP14].

<sup>5</sup>The PCMark benchmark tool for performance includes benchmarking and measurement parts which is the perfect combination of efficiency and performance. It is a complete PC performance benchmark for home and business[Cor14a].

the different VMs. The summation of the values will be compared with the measured value from the power meter. The values will be written in a text file which will be formatted afterwards. The formatted values are used to analyze the test result. One instrument that supports the analyzing process is Matlab. It is utilized to plot the solution values. Plotting the test result values make it visible and easier to see where and how big the error is.

### 6.5.6 Test Results

To measure the accuracy of the system, the mean value  $\bar{x}$  and the standard deviation  $S$  are calculated with the test results. The mean value  $\bar{x}$ , uses a vector  $A$  with dimension  $N$  as input. The mean value is identified in the equation as follows:

$$\bar{x} = \frac{1}{N} \sum_{i=1}^N A_i \quad (6.2)$$

The standard deviation  $S$  is identified in the equation as follows:

$$S = \sqrt{\frac{1}{N} \sum_{i=1}^N (A_i - \bar{x})^2} \quad (6.3)$$

Table 6.6 shows the results of the different scenarios and their figures.

<i>Scenario</i>	<i>Mean Error</i>	<i>Standard Deviation</i>	<i>Figures</i>
Scenario I	-0.1951 %	6.564 Watt	Figure 6.4
Scenario II	-4.3124 %	13.208 Watt	Figure 6.5
Scenario III	-4.8909 %	15.886 Watt	Figure 6.6
Scenario IV	-4.5235 %	15.122 Watt	Figure 6.7
Scenario V	-4.7778 %	16.806 Watt	Figure 6.8
Scenario VI	-2.3355 %	17.181 Watt	Figure 6.9
Scenario VII	-1.4245 %	14.604 Watt	Figure 6.10
Scenario VIII	1.9501 %	2.1545 Watt	Figure 6.11

Table 6.6: Results of the different Test Scenarios with Average Error and Figures

The mean error of all scenarios ranges between -4.7877 percent to 1.9501 percent. The error falls further into the negative values if the number of cores, VMs and activity increases. If there are more VMs with a higher number of cores, which are running in idle mode, the error is positive. These tests show that the system is very accurate for VMs that only have one core allocated but the error increases if there are more cores allocated. In addition, the error is always in the plus, minus five percent range which was one of the main requirements.

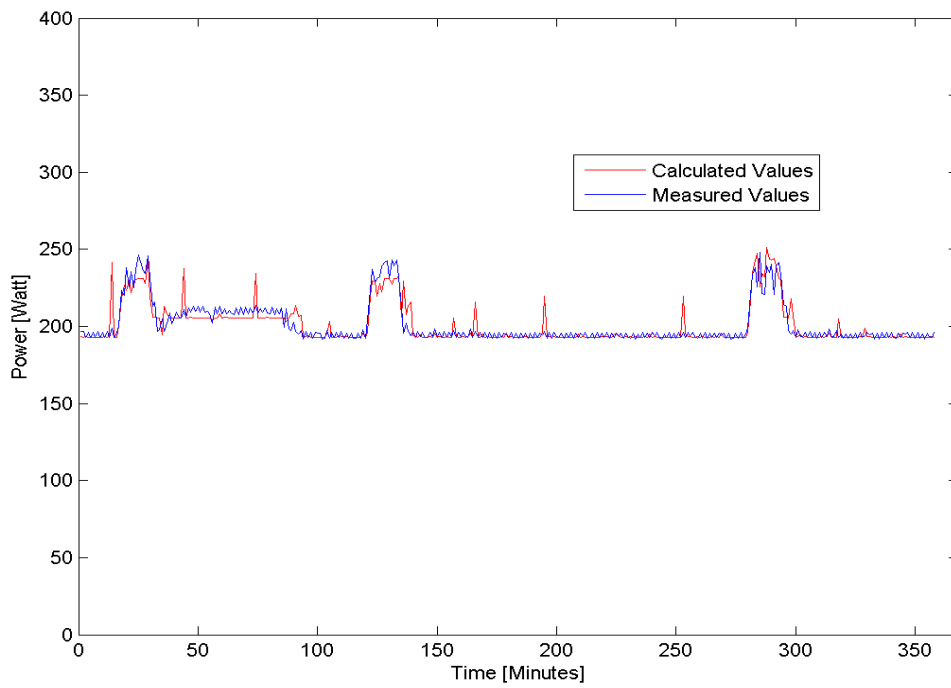


Figure 6.4: Results of Test Scenario I. Table 6.5

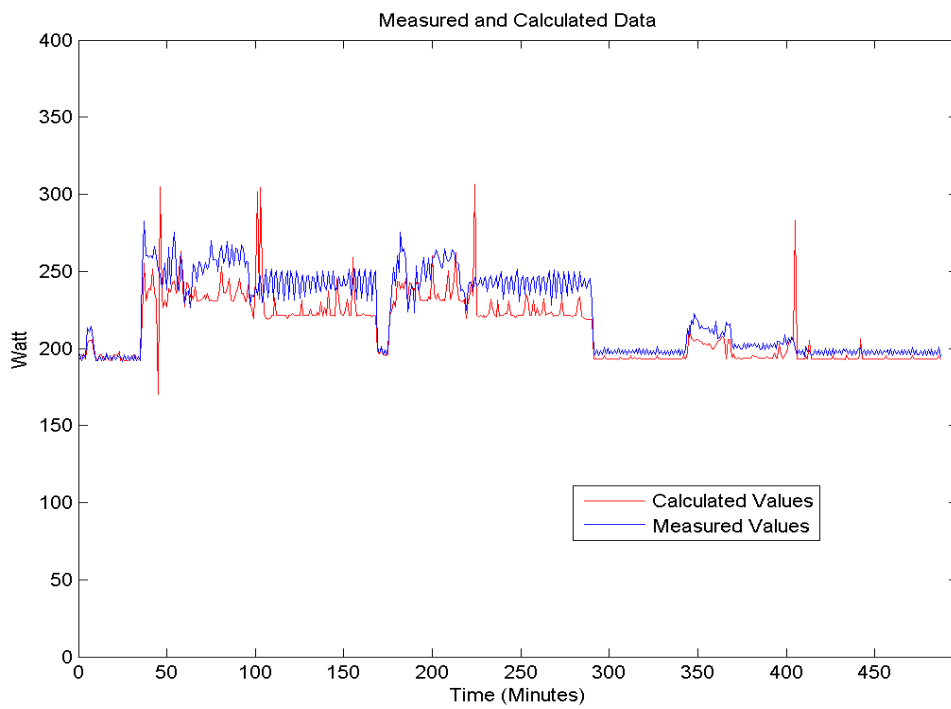


Figure 6.5: Results of Test Scenario II. Table 6.5

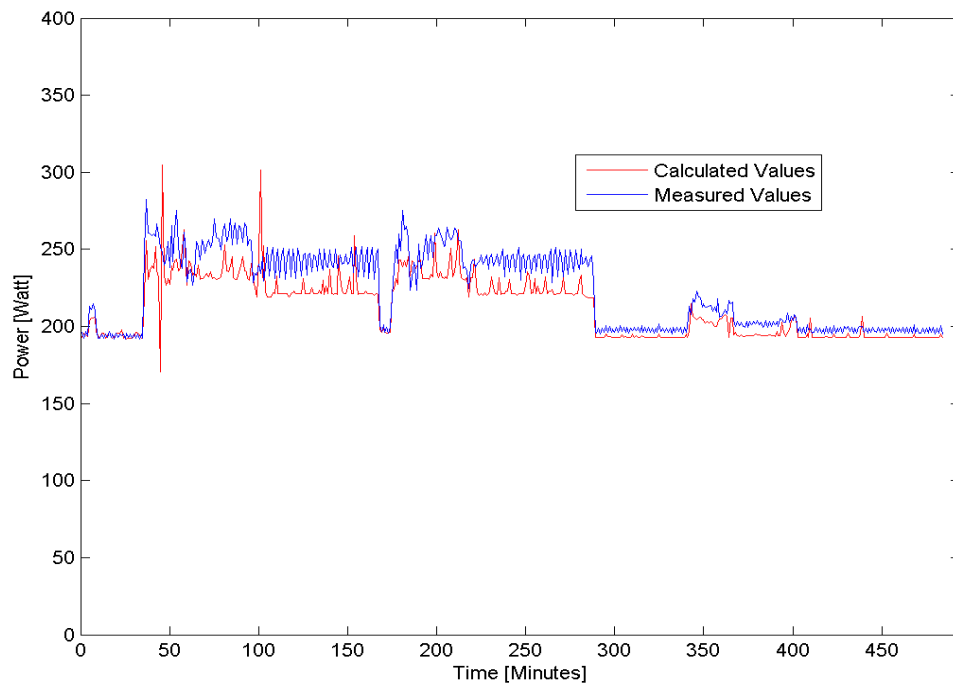


Figure 6.6: Results of Test Scenario III. Table 6.5

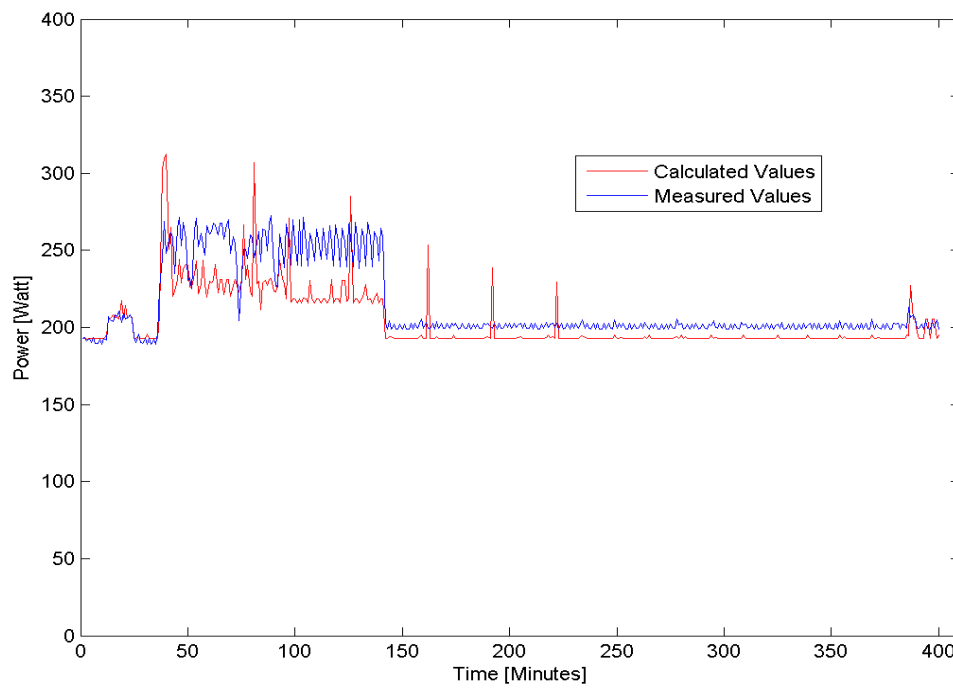


Figure 6.7: Results of Test Scenario IV. Table 6.5



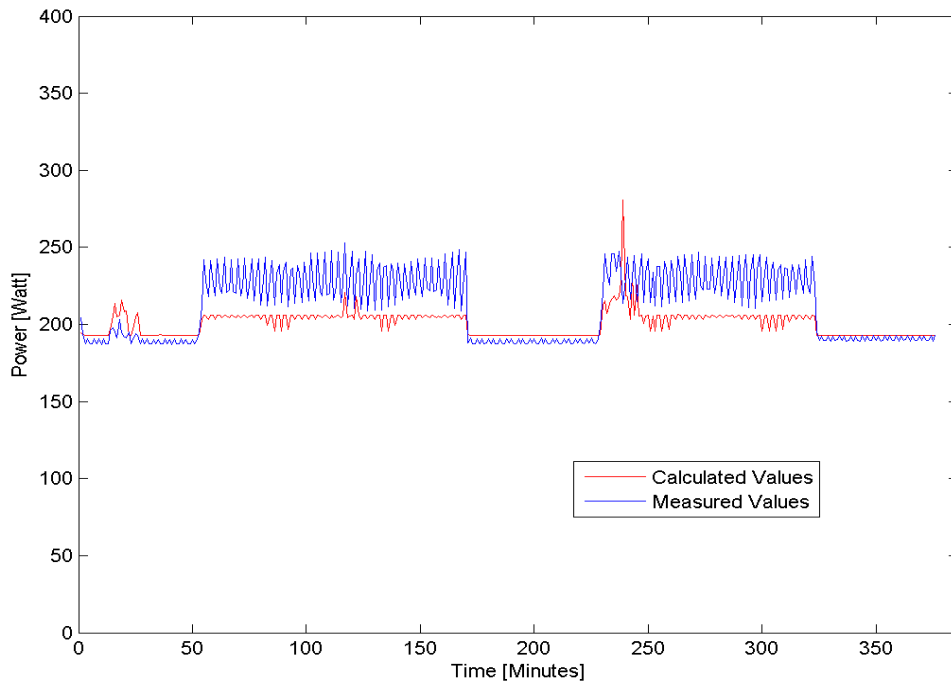


Figure 6.8: Results of Test Scenario V. Table 6.5

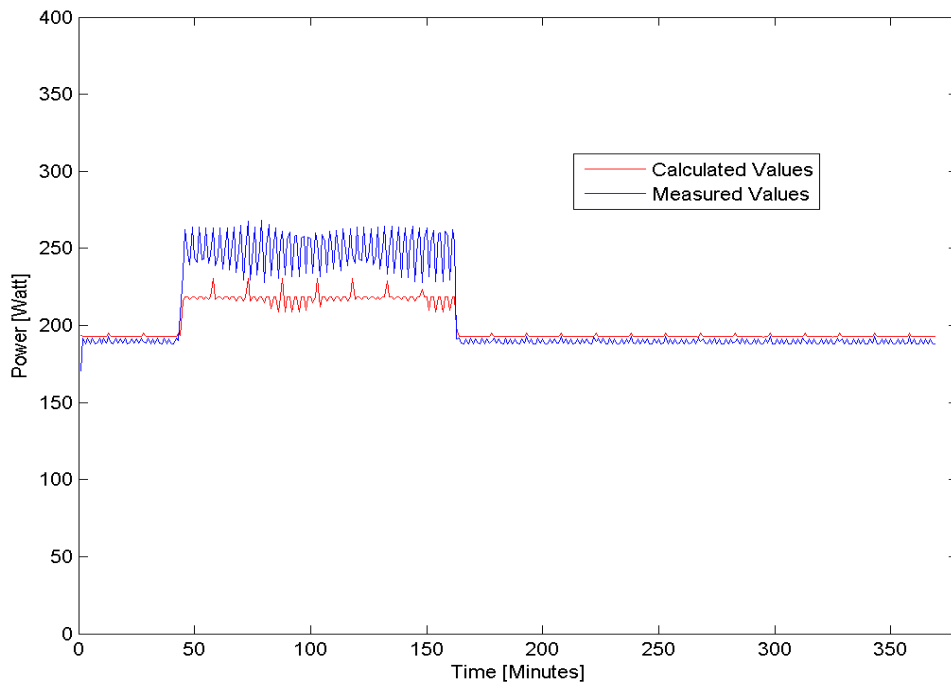


Figure 6.9: Results of Test Scenario VI. Table 6.5

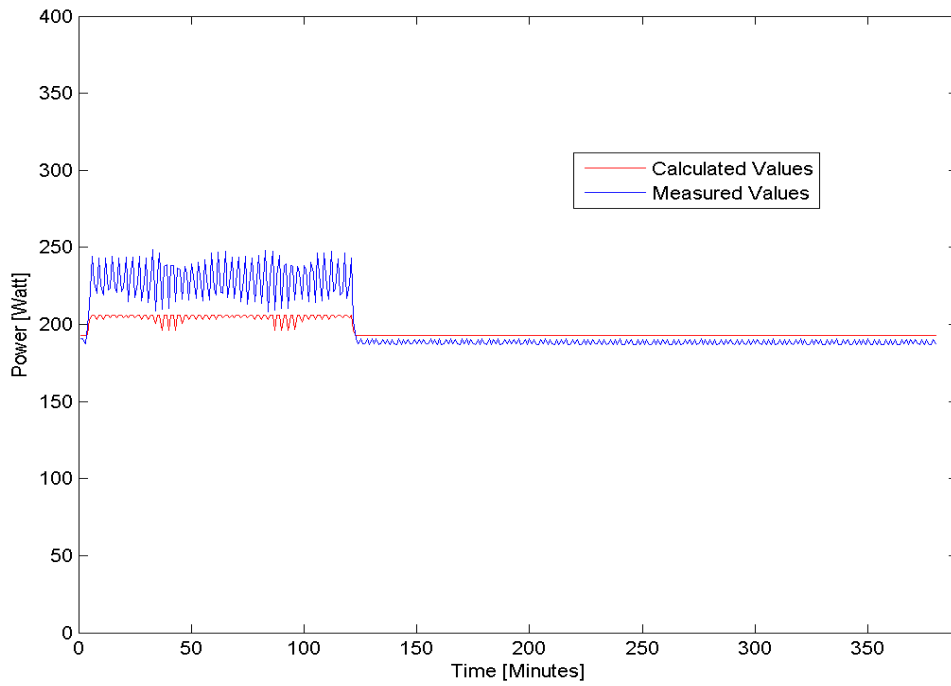


Figure 6.10: Results of Test Scenario VII. Table 6.5

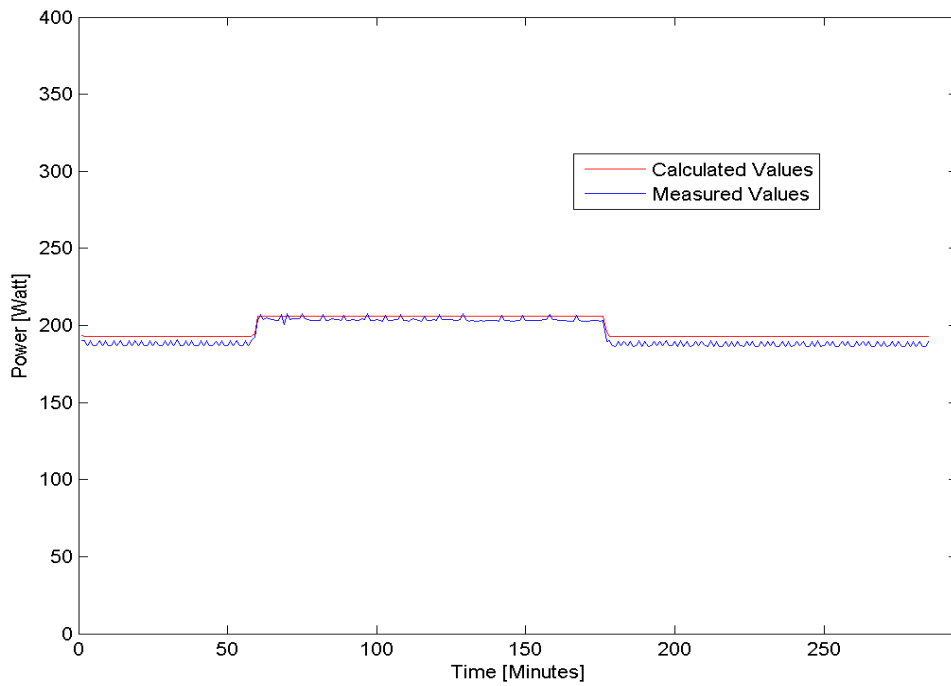


Figure 6.11: Results of Test Scenario VIII. Table 6.5

## 6.6 Papillon Master Stress Test

The reason behind the stress test on the Papillon Master was to figure out how many Papillon Clients can report to one Papillon Master. An application was developed for this stress test which simulates clients. These clients are sending dummy data to the master. The master response is either okay on success or error if something went wrong. Dummy values are used because of performance reasons. To calculate the real system information for this amount of simulated clients would be to much effort. The test application is implemented in Java using the NetBeans IDE. There are different settings for the test, which can be set in the test application. The different parameter are described as below:

- **Number of Clients** is the number of clients that the testing tool should be simulate.
- **IP Address of the Master** this is the IP address of the Papillon Master server which should be tested.
- **Port** is the port number of the tested Papillon Master server.
- **Offset** displays the offset where the host id starts. This value can be obtained from the Papillon Master's database or the internal Papillon GUI.

The main goals of these tests is, to find the maximum number of clients, that can report at the same time. In addition, to get the response time, delay time and the amount of data sent.

### 6.6.1 Papillon Master Stress Test Setup

This test will be done on two different servers. One will be the master and the other a laptop where the stress application is running. The used hardware is detailed as follows:

- **Test PC: Lenovo T430s**
  - **Operating System:** Microsoft Windows 8.1 Professional
  - **Processor:** Intel Core i5-3320M CPU 2.60GHz
  - **RAM:** 8 GB
  - **OS Architecture:** 64-bit
- **Server I: IBM xSeries 336**
  - **Operating System:** Ubuntu 12.04 LTS
  - **Processor:** Intel Xeon 3.2 GHz

- **RAM:** 4 GB
- **OS Architecture:** 64-bit
- **Server II: Dell PowerEdge SC1425**
  - **Operating System:** Ubuntu 12.04 LTS
  - **Processor:** Intel Xeon 2.8 GHz
  - **RAM:** 4 GB
  - **OS Architecture:** 64-bit Operating System

### 6.6.2 Papillon Master Stress Test Results

Tables 6.7 and 6.8 shows the result of the stress test from these two servers.

<i>Server</i>	<i>Operating System</i>	<i>No. Clients</i>	<i>Test Time</i>	<i>Av. Response Time</i>	<i>Fail</i>
Server I	Ubuntu 14.04	1000	21066 ms	1954 ms	0
Server I	Ubuntu 14.04	1200	21362 ms	2440 ms	0
Server I	Ubuntu 14.04	1500	32624 ms	3404 ms	0
Server I	Ubuntu 14.04	3000	56881 ms	5544 ms	0
Server I	Ubuntu 14.04	5000	57285 ms	5669 ms	626
Server II	Ubuntu 12.04	1000	20032 ms	2074 ms	0
Server II	Ubuntu 12.04	1200	20670 ms	4398 ms	0
Server II	Ubuntu 12.04	1500	30279 ms	4312 ms	0
Server II	Ubuntu 12.04	3000	59598 ms	3114 ms	212

Table 6.7: Table with the first part of the Stress Test Results from Server I and Server II

<i>Server</i>	<i>No. Clients</i>	<i>Min. Response Time</i>	<i>Max. Response Time</i>	<i>KB Sent</i>
Server I	1000	47 ms	6739 ms	460kB
Server I	1200	94 ms	11089 ms	552kB
Server I	1500	97 ms	8908 ms	695kB
Server I	3000	46 ms	1750 ms	1383kB
Server I	5000	63 ms	36058 ms	2348kB
Server II	1000	107 ms	11388 ms	460kB
Server II	1200	172 ms	10234 ms	552kB
Server II	1500	188 ms	11403 ms	690kB
Server II	3000	203 ms	16006 ms	1412kB

Table 6.8: Table with the second part of the Stress Test Results from Server I and Server II

These results show that the number of clients, which a master can handle, depends on

the used hardware. Server I could handle 3000 clients without problem and server II could only handle 1500 clients without errors.

## 6.7 Import Data Center via XML File

This test should determine the maximal number of clients that can be installed using an XML file. This is very important, since we need this for future customer use. A python script was developed to create an XML file, which can be imported. This created file follows the XML definition and structure from UCD. The Python script enables us to automatically generate a data center XML file with a specified number of hosts.

The import of a DC is completed using the internal Papillon GUI. Figure 6.12 displays the internal GUI data center import view using Google Chrome<sup>6</sup>.

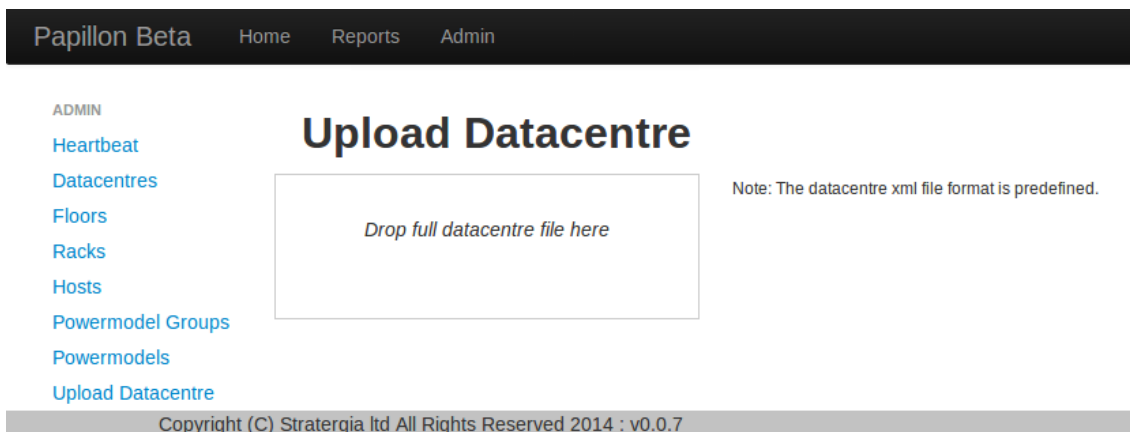


Figure 6.12: Import Data Center with Internal Papillon GUI

### 6.7.1 Conclusion of the Import Data Center via XML File Test

After a successful import, the message "Import Successful" is displayed otherwise an error message is presented. This test scenario was with **60000 hosts**. The only required action that had to be taken before the test successfully finished was to disable the maximum file size in Apache Tomcat. Apache Tomcat has a file size limit set by default.

The import of 60000 hosts needed more than ten minutes and was successful! It was verified with the Papillon Master database.

---

<sup>6</sup><https://www.google.com/chrome>

## 6.8 Tool to identify the Number of Possible Virtual Machines for Given Server Setup

It is difficult to know how many VMs can run on a physical server. Not only the hypervisor limits the number of VMs running on a server, it is also the hardware which creates another limit. This tool was created to determine the maximum number of possible VMs running on a physical server. The tool was developed in Java including a GUI, to make the usage more easy. This tool allows the user to set the physical server configuration parameters as describes as follows:

- Number of virtual Processors
- Memory in GB
- Hard Drive in GB

First, the server hardware parameter has to be entered carefully into the tool so that afterwards the calculation can be performed. If the user is interested to save the calculation is this possible with the save button. The tool calculates the maximum number of VMs, using a Linux or a Windows based OS, and show the main ground for the limitation. This implies, if the limitation ground is for example the memory, more memory on the physical server is needed, so then there could be more VMs being added. This tool gives recommendations, but the resulting number of VMs can be different, according to their usage. Figure 6.13 shows the tool with sample output.

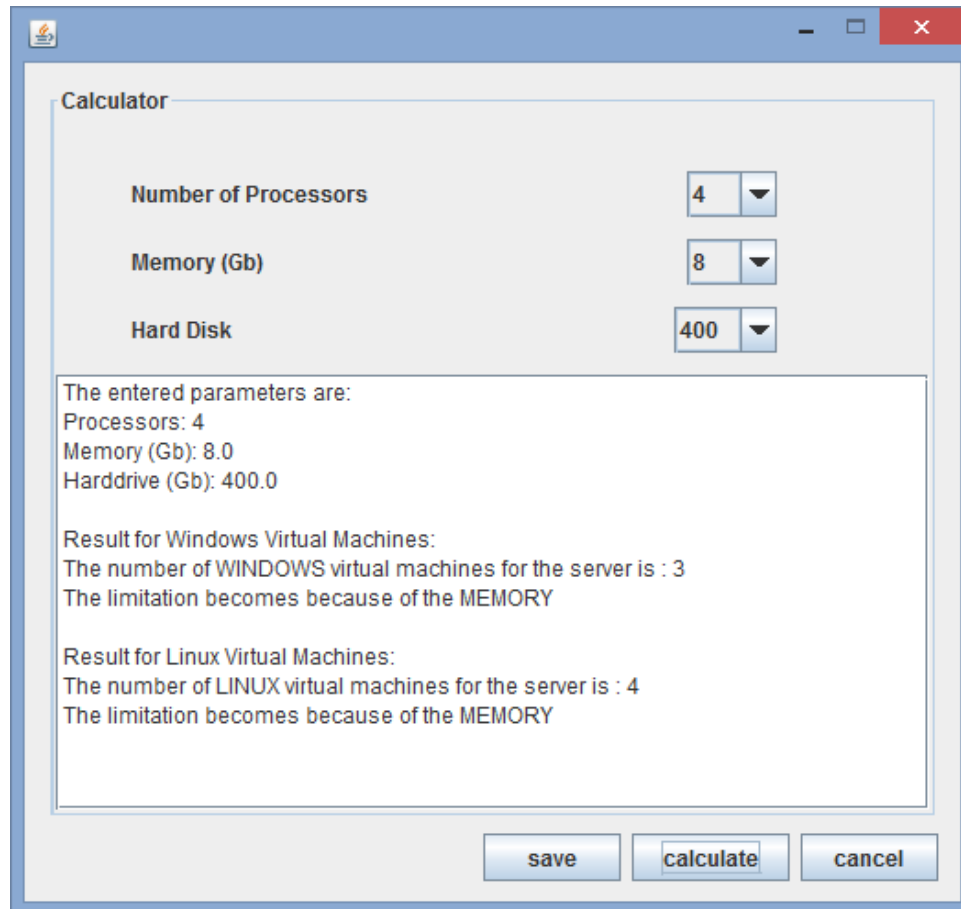


Figure 6.13: Tool to get the Maximum Number of Possible VMs on a Physical Server

## Chapter 7

# Conclusion and Future Work

### 7.1 Conclusion

The **Profiling and Auditing of Power Information in Large and Local Organizational Networks (PAPILLON)** system is a great solution that can make IT become "Greener". When Papillon was used in a physical environment, it was already a good solution that provided an accurate overview of the power consumption in a DC.

One big step forward was the development of the VM support because virtualization becomes more and more important.

The accuracy of the estimated power for VMs is high. The mean error is within the range of -5% to 2% which is good for VMs compared to the related work [RRK08, KZL<sup>+</sup>10, LWYG12, et.al.]. The accuracy depends on the number of VMs running on the server and on the setup of the VMs and their workload.

Nevertheless, I believe that we are able to further increase the accuracy even if it is already quite high and especially when we take the power model and its generation into consideration.

### 7.2 Future Work

The newly implemented field in the Papillon Master which displays the number of running VMs on the physical server is a static number. A dynamic value can be used for future work. One way to realize and implement the dynamic value is to have the Papillon Master recognize it after aVM sends data to the master, which is the physical server the VM is running on. This can be served with an additional flag in the Papillon



database. This flag is a unique ID which is set in the beginning. If the host is a VM, the unique number of the physical server has to be entered.

The master can automatically change the number of running VMs on the specific physical host or server. If the VM is not sending data for ten minutes, the number of running VMs can be decreased by one because the VM is no longer responding. After the VM is sending data again, the number of running VMs can be decreased again. It should not be possible to have zero VMs running on the server because the master would not be able to handle the null value in the consumed power calculation.

Furthermore, the DCs Storage Area Network (SAN) and Network Attached Storage (NAS) are common technologies which are not included in the power estimation from the Papillon Master. For this very reason, including the SAN and NAS systems in the power estimation would be one more step forward. It would be even better if the SAN and NAS system devices would have their own power profile or if the power profile could be included in the already existing power model and power model generation process.

It would also be interesting to see how accurate the Papillon system is by using a hosted hypervisor such as Oracle VirtualBox to compare with the native hypervisor VMware ESXi's results which were used in this work.

The Papillon system should be more flexible together and consistent in the way the system information is retrieved from the different OSs. This means that the SIGAR library should be used for all supported OSs.

If the Papillon system only uses the SIGAR library to get the system information parameters, Papillon would be able to support more OSs. The SIGAR library is a very powerful library which supports a huge variety of OSs. This is why it would benefit Papillon to make use of the advantages the SIGAR library has to offer.

# Abbreviations

<b>AC</b>	Alternating Current
<b>ACD</b>	Apache Commons Daemons
<b>AJAX</b>	Asynchronous JavaScript and Extensible Markup Language
<b>API</b>	Application Programming Interface
<b>BSD</b>	Berkeley Software Distribution
<b>CF</b>	Carbon Footprint
<b>CPU</b>	Central Processing Unit
<b>CRAC</b>	Computer Room Air Conditioning Unit
<b>CRAH</b>	Computer Room Air Handling Unit
<b>CSS</b>	Cascadian Style Sheet
<b>DC</b>	Data Center
<b>DCIM</b>	Data Center Infrastructure Management
<b>DLL</b>	Dynamic Linked Library
<b>DUT</b>	Device Under Test
<b>GC</b>	Garbage Collection
<b>GPL</b>	General Public License
<b>GUI</b>	Graphical User Interface
<b>HTML</b>	Hypertext Markup Language
<b>IDE</b>	integrated development environment
<b>I/O</b>	Input/Output
<b>IP</b>	Internet Protocol
<b>IT</b>	Information Technology

<b>JNA</b>	Java Native Access
<b>JSON</b>	JavaScript Object Notation
<b>JSP</b>	Java Server Page
<b>JSW</b>	Java Service Wrapper
<b>JVM</b>	Java Virtual Machine
<b>KVM</b>	Kernel Based Virtual Machine
<b>L4J</b>	Launch4j
<b>LGPL</b>	Lesser General Public License
<b>NAS</b>	Network Attached Storage
<b>OS</b>	Operating System
<b>OSHI</b>	Operating System and Hardware Information
<b>OOP</b>	Object Oriented Programming
<b>PAPILLON</b>	Profiling and Auditing of Power Information in Large and Local Organizational Networks
<b>PDU</b>	Power Distribution Unit
<b>PID</b>	Process Identification Number
<b>PSFL</b>	Python Software Foundation License
<b>PUE</b>	Power Usage Effectiveness
<b>RAM</b>	Random Access Memory
<b>REST</b>	Representational State Transfer
<b>SAN</b>	Storage Area Network
<b>SIGAR</b>	System Information Gatherer and Reporter
<b>SQL</b>	Structured Query Language
<b>UNIX</b>	Uniplexed Information and Computing Service
<b>VM</b>	Virtual Machine
<b>VMM</b>	Virtual Machine Monitor
<b>XML</b>	Extensible Markup Language
<b>XSLT</b>	Extensible Style-sheet Language
<b>YAJSW</b>	Yet Another Java Service Wrapper

# Appendix A

## Visualization of the NetBeans Profiler Results

### A.1 Description

This section contains figures of the NetBeans Profiler tool. Each figure is a live profiling result which is created on different OSs. The **VM Telemetry Overview** window in the tool shows the live results of the profiling task.

The profiler tool provided by NetBeans allows users to save results or capture states during each profiling session. All monitoring process information is displayed in the NetBeans IDE. Furthermore, there are a lot of different monitoring settings available which the user can use for different profiling options such as for memory monitoring, CPU monitoring and thread activity [Ora13].

Each figure consists of three different diagrams which are created from the live monitoring process. These are the smaller versions of the graphs which are available in their own view within the tool. The diagrams showing the Memory (Heap), the Garbage Collection (GC) and the Threads/Loaded Classes are explained in detail below [Ora13]:

- **Memory (Heap)** is shown in the first diagram and displays the total and used size of the heap.
- **Memory GC** is shown in the second diagram and displays the relative time that was spent in garbage collection during the profiling session and the number of surviving generations.
- **Threads/Loaded Classes** is shown in the third diagram and displays the total number of threads and the number of threads in the profiled application Java

Virtual Machine (JVM).

Furthermore, if the user hovers over one of the graphs, a tooltip is displayed and provides more detailed data for the given point which is shown in figures A.1, A.2, A.3 and A.4. Each of the three diagrams can be opened in separate views. These separate views enable more functionality for profiling, monitoring, performing different analyses and viewing more detailed information.

Furthermore, the CPU activity is also profiled but unfortunately not shown in the Telemetry Overview. During the profiling session, NetBeans Profiler allows users to take a snapshot, save all data or export both snapshots and data at any given time.

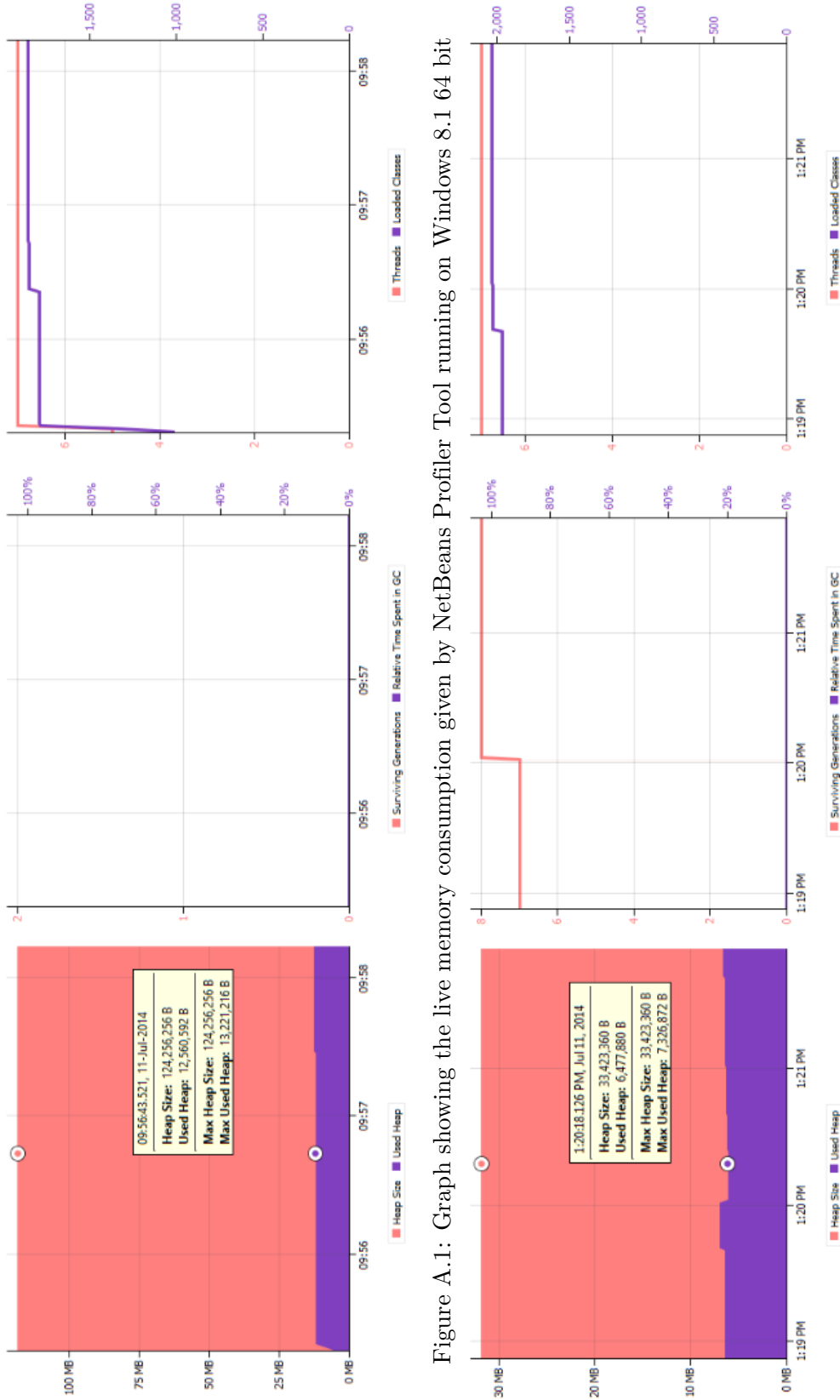


Figure A.1: Graph showing the live memory consumption given by NetBeans Profiler Tool running on Windows 8.1 64 bit

Figure A.2: Graph showing the live memory consumption given by NetBeans Profiler Tool running on Windows 7 64 bit

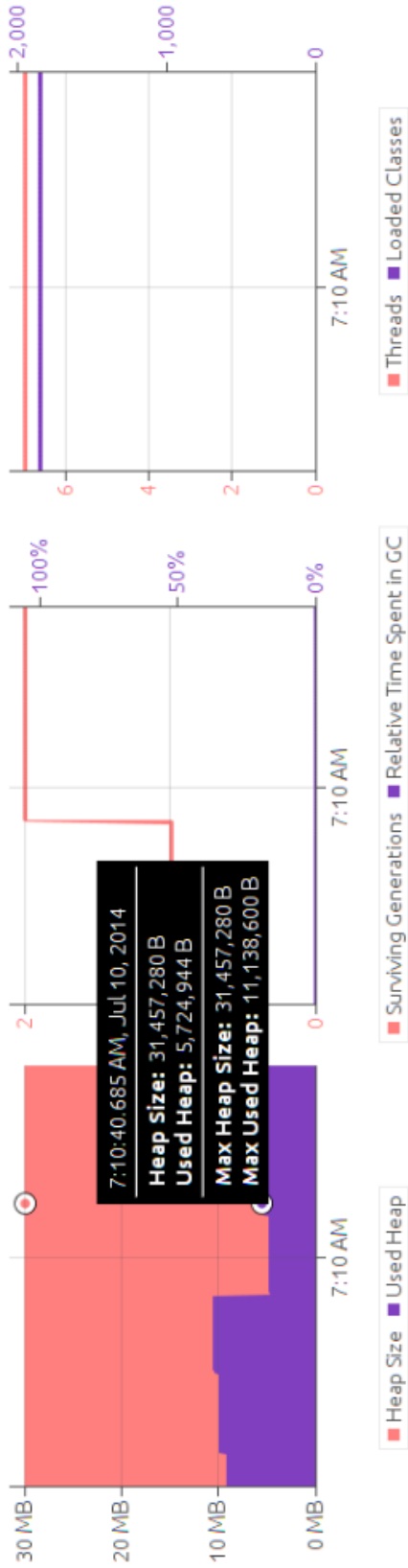


Figure A.3: Graph showing the live memory consumption given by NetBeans Profiler Tool running on Ubuntu 14.04 64 bit

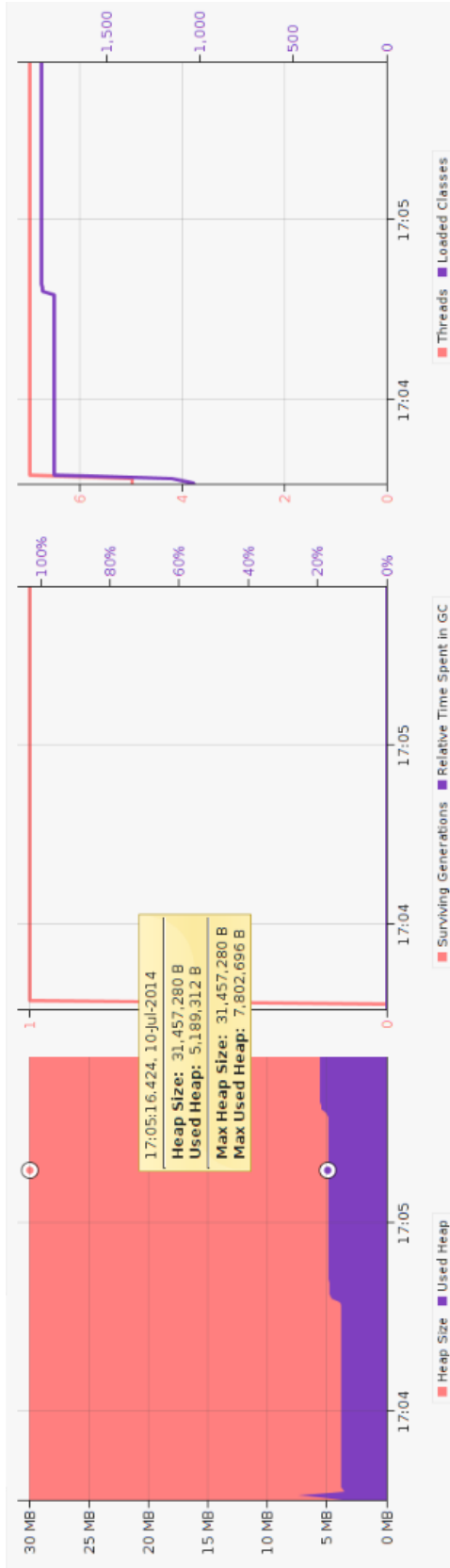


Figure A.4: Graph showing the live memory consumption given by NetBeans Profiler Tool running on Mint 17 Linux 64 bit

## Appendix B

# Client Installer Scripts Source Code

This section shows the source code which is explained in section 5.4.

```
----- getCorrectPid.bat -----  
1  echo off  
2  REM Description:  
3  REM      getCorrectPid.bat  
4  REM      This script is called during the installation process.  
5  REM  
6  REM      Author - Andreas Abraham  
7  REM      Version - 1.0  
8  REM  
9  REM      Warning: Please don't change!  
10  
11 java -jar phy-client-v2.0.jar masterIP=%1 port=%2 hostID=%3
```

Listing 1: The Source Code from the getCorrectPid.bat Script File



```

----- setup.bat -----
1  echo off
2  REM Description:
3  REM Setup script to start the installation routine
4  REM The cmd line arguments are checked here
5  REM After success the powershell script is called
6  REM
7  REM     Author - Andreas Abraham
8  REM     Version - 1.0
9  REM
10 REM     Usage: call scrip as admin as follow:
11 REM     setup.bat -ip <IP Master> -po <Port Master> -id <Host ID>
12
13
14 REM Check the arguments
15 IF NOT "%1" == "-ip" GOTO Error
16 IF     "%2" == ""    GOTO Error
17 IF NOT "%3" == "-po" GOTO Error
18 IF     "%4" == ""    GOTO Error
19 IF NOT "%5" == "-id" GOTO Error
20 IF     "%6" == ""    GOTO Error
21
22 GOTO Continue
23
24 :Continue
25 REM Call Powershell with script execution enabled
26     Powershell -executionpolicy unrestricted ^
27                 -File installPapillonClient.ps1 %2 %4 %6
28 GOTO End
29
30 :Error
31 REM Print Error and Correct Argument usage Message
32     echo "***** Wrong Argument usage! *****"
33     echo "* -ip <IP Address> -po <Port> -id <Host ID> *"
34     echo "*****"
35
36 :End

```

Listing 2: The Source Code from the setup.bat Script File

```

----- installPapillonClient.ps1 -----
1  #####
2  # Install Papillon Client Powershell Script #
3  #   Version: 1.0 #
4  #   Author: Andreas Abraham #
5  #   This Powershell Script is used for the #
6  #   installation process and called from setup.bat #
7  #####
8
9  IF ($args.Length -ne 3) {
10     echo "Error wrong Commandline Argument usage!"
11     return;
12 }
13 ELSE {
14     $ip = $args[0]
15     $po = $args[1]
16     $id = $args[2]
17 }
18
19 $arguments = $ip + " " + $po + " " + $id
20 $batFile = "getCorrectPid.bat"
21
22 Start-Process $batFile $arguments
23
24 Sleep -s 2
25
26 $javaProcess = Get-Process java
27 echo $javaProcess.Id
28
29 $process = Start-Process ..\yajsw-stable-11.11\bat\genConfig.bat |
30             $javaProcess.Id -PassThru -NoNewWindow
31 $process.WaitForExit()
32
33 Stop-Process $javaProcess.Id
34
35 $instProc = Start-Process ..\yajsw-stable-11.11\bat\installService.bat |
36             -PassThru -NoNewWindow
37 $instProc.WaitForExit()
38
39 $startProc = Start-Process ..\yajsw-stable-11.11\bat\startService.bat |
40             -PassThru -NoNewWindow
41 $startProc.WaitForExit()

```

Listing 3: The Source Code from the installPapillonClient.ps1 Script File

# Bibliography

- [AAH<sup>+</sup>03] I. Ahmad, J.M. Anderson, A.M. Holler, R. Kambo, and V. Makhija. An analysis of disk performance in vmware esx server virtual machines. In *Workload Characterization, 2003. WWC-6. 2003 IEEE International Workshop on*, pages 65–76, Oct 2003.
- [BC10] AE.H. Bohra and V. Chaudhary. Vmeter: Power modelling for virtualized clouds. In *Parallel Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010 IEEE International Symposium on*, pages 1–8, April 2010.
- [Cor14a] Futuremark Corporation. PC Mark Benchmark, 2014. Accessed at: July 2014, <http://www.futuremark.com/benchmarks/pcmark>.
- [Cor14b] Oracle Corporation. NetBeans, 2014. Accessed at: July 2014, <https://netbeans.org/>.
- [COR14c] YOKOGAWA ELECTRIC CORPORATION. The Green Grid, 2014. Accessed at: July 2014, [http://tmi.yokogawa.com/discontinued-products/digital-power-analyzers/digital-power-analyzers/wt210wt230-digital-power-meters/#tm-wt210\\_01.htm](http://tmi.yokogawa.com/discontinued-products/digital-power-analyzers/digital-power-analyzers/wt210wt230-digital-power-meters/#tm-wt210_01.htm).
- [CSK09] *Some Joules Are More Precious Than Others: Managing Renewable Energy in the Datacenter*, This work was supported in part by NSF CAREER Award CCF-0448413, grants CNS-0615045, CCF-0621472, and by an IBM Faculty Award. ACM, 2009.
- [CXY<sup>+</sup>13] Wen Chengjian, Long Xiang, Yang Yang, Fan Ni, and Yifen Mu. System power model and virtual machine power metering for cloud computing pricing. In *Intelligent System Design and Engineering Applications (ISDEA), 2013 Third International Conference on*, pages 1379–1382, Jan 2013.
- [DA14] Andrew Donoghue and Rhonda Ascierio. Strategia develops DCIM for European colos to compete via IT energy transparency. Technical report, 451 Research, August 2014.
- [DCI14] *Data Center Infrastructure Management, Market Overview and Orientation Guide*. White Paper eco Datacenter Expert Group, May 2014.

- [DMR10] G. Dhiman, K. Mihic, and T. Rosing. A system for online power prediction in virtualized environments using gaussian mixture models. In *Design Automation Conference (DAC), 2010 47th ACM/IEEE*, pages 807–812, June 2010.
- [EA13] A. Elsayed and N. Abdelbaki. Performance evaluation and comparison of the top market virtualization hypervisors. In *Computer Engineering Systems (ICCES), 2013 8th International Conference on*, pages 45–50, Nov 2013.
- [Fie00] Roy Thomas Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, UNIVERSITY OF CALIFORNIA, IRVINE, 2000.
- [Fou14] The Apache Software Foundation. Apache Tomcat, 2014. Accessed at: August 2014, <http://tomcat.apache.org/>.
- [GGP12] *PUE: A COMPREHENSIVE EXAMINATION OF THE METRIC*, White Paper 49. The Green Grid, 2012. Accessed at: June 2014, [http://www.thegreengrid.org/~media/WhitePapers/WP49-PUE%20A%20Comprehensive%20Examination%20of%20the%20Metric\\_v6.pdf?lang=en](http://www.thegreengrid.org/~media/WhitePapers/WP49-PUE%20A%20Comprehensive%20Examination%20of%20the%20Metric_v6.pdf?lang=en).
- [GHJ14] C. Gu, H. Huang, and X. Jia. Power metering for virtual machine in cloud computing-challenges and opportunities. *Access, IEEE*, 2:1106–1116, 2014.
- [GNM09] *Green Networking for Major Components of Information Communication Technology Systems*, EURASIP Journal on Wireless Communications and Networking. Hindawi Publishing Corp. New York, NY, United States, September 2009.
- [GPH12] *How Clean is Your Cloud?*, Make IT Green. Greenpeace International, Ottho Heldringstraat 5, 1066 AZ Amsterdam, Netherlands, April 2012. Accessed at: June 2014, <http://www.greenpeace.org/international/en/publications/Campaign-reports/Climate-Reports/How-Clean-is-Your-Cloud>.
- [Gri14] The Green Grid. The Green Grid, 2014. Accessed at: June 2014, <http://www.thegreengrid.org>.
- [Gro02] W3C’s Technical Architecture Group. Architectural Principles of the World Wide Web, 2002. Accessed at: December 2014, <http://www.w3.org/TR/2002/WD-webarch-20020830>.
- [Gro14] W3C’s Technical Architecture Group. XSL Transformations (XSLT) Version 3.0, 2014. Accessed at: December 2014, <http://www.w3.org/TR/xslt-30>.
- [Hyp14] Hyperic. Hyperic SIGAR API - One API to access system information

- regardless of the underlying platform, 2014. Accessed at: Mai 2014, <http://www.hyperic.com/products/sigar>.
- [ICT14] *ICT Facts and Figures*, The World in 2014. International Telecommunication Union, April 2014. Accessed at: June 2014, <http://www.itu.int/en/ITU-D/Statistics/Documents/facts/ICTFactsFigures2014-e.pdf>.
- [Inc14a] VMware Inc. How a hypervisor-converged software-defined data center enables a better private cloud. *White Paper*, jun 2014.
- [Inc14b] VMware Inc. VMware, 2014. Accessed at: August 2014, <http://vmware.com>.
- [Inc15] VMware Inc. VMware vSphere, 2015. Accessed at: March 2015, <http://www.vmware.com/products/vsphere/>.
- [JGK08] Ph.D. Jonathan G. Koomey. Worldwide electricity used in data centers. *Lawrence Berkeley National Laboratory, USA and Stanford University, Oakland, USA*, September 2008. Accessed at: June 2014, <http://www.koomey.com/research.html>.
- [JGK11] Ph.D. Jonathan G. Koomey. GROWTH IN DATA CENTER ELECTRICITY USE 2005 TO 2010. *A report by Analytics Press, completed at the request of The New York Times*, August 2011. Accessed at: June 2014, <http://www.koomey.com/research.html>.
- [JUn14] JUnit. JUnit is a simple framework to write repeatable tests. It is an instance of the xUnit architecture for unit testing frameworks., 2014. Accessed at: February 2015, <http://junit.org/>.
- [Kam13] Bernd Kampl. Characterization and Modelling of Server Energy Consumption. Master's thesis, Institute for Technical Informatics, Graz University of Technology, 2013.
- [KP14] Mika Kuoppala and Peter Palfrader. Tiotest, 2014. Accessed at: July 2014, <http://linux.die.net/man/1/tiotest>.
- [KT04] R. Khare and R.N. Taylor. Extending the representational state transfer (rest) architectural style for decentralized systems. In *Software Engineering, 2004. ICSE 2004. Proceedings. 26th International Conference on*, pages 428–437, May 2004.
- [KZL<sup>+</sup>10] Aman Kansal, Feng Zhao, Jie Liu, Nupur Kothari, and Arka A. Bhattacharya. Virtual machine power metering and provisioning. In *Proceedings of the 1st ACM Symposium on Cloud Computing*, SoCC '10, pages 39–50, New York, NY, USA, 2010. ACM.
- [Lam10] Frank Lampe. *Green-IT, Virtualisierung und Thin Clients*. Mit neuen IT-

Technologien Energieeffizienz erreichen, die Umwelt schonen und Kosten sparen. Vieweg+Teubner | GWV Fachverlage GmbH, 2010.

- [Li10] Peng Li. Centralized and decentralized lab approaches based on different virtualization models. *J. Comput. Sci. Coll.*, 26(2):263–269, dec 2010.
- [Ltd15] Stratergia Ltd. Stratergia Data Centre Energy/Power Audit Programme, 2015. Accessed at: March 2015, <http://stratergia.com/papillon/stratergia-data-centre-energypower-audit-programme/>.
- [LWYG12] Yanfei Li, Ying Wang, Bo Yin, and Lu Guan. An online power metering model for cloud environment. In *Network Computing and Applications (NCA), 2012 11th IEEE International Symposium on*, pages 175–180, Aug 2012.
- [MJZ<sup>+</sup>13] Chongya Ma, Zhiying Jiang, Ke Zhang, Guangfei Zhang, Zhixiong Jiang, Chunyang Lu, and Yushan Cai. Virtual machine power metering and its applications. In *Global High Tech Congress on Electronics (GHTCE), 2013 IEEE*, pages 153–156, Nov 2013.
- [MPK11] P. Muditha Perera and Chamath Keppitiyagama. A performance comparison of hypervisors. In *Advances in ICT for Emerging Regions (ICTer), 2011 International Conference on*, pages 120–120, Sept 2011.
- [mpp] Linux man-pages project. Linux Programmer’s Manual - proc - process information pseudo-filesystem. Accessed at: Mai 2014, <http://man7.org/linux/man-pages/man5/proc.5.html>.
- [Ora13] Oracle. Testing and Profiling Java Application Projects, 2013. Accessed at: February 2015, [http://docs.oracle.com/cd/E40938\\_01/doc.74/e40142/test\\_profile\\_japps.htm#BABIGDGA](http://docs.oracle.com/cd/E40938_01/doc.74/e40142/test_profile_japps.htm#BABIGDGA).
- [Ora15] Oracle. VirtualBox, 2015. Accessed at: March 2015, <https://www.virtualbox.org>.
- [OSH14] OSHI. Operating System and Hardware Information OSHI, 2014. Accessed at: Mai 2014.
- [RES13] *Highly Energy-Efficient Datacenters in Practice*, Case Studies of Advanced Innovations in Datacenter Technologies and Techniques, 2012-2013. 451 Research, 2013.
- [RRK08] Suzanne Rivoire, Parthasarathy Ranganathan, and Christos Kozyrakis. A comparison of high-level full-system power models. In *Proceedings of the 2008 Conference on Power Aware Computing and Systems*, HotPower’08, pages 3–3, Berkeley, CA, USA, 2008. USENIX Association.

- [sou14] sourceforge. Yet Another Java Service Wrapper, 2014. Accessed at: August 2014, <http://yajsw.sourceforge.net>.
- [SS14] S.S. Sahasrabudhe and S.S. Sonawani. Comparing openstack and vmware. In *Advances in Electronics, Computers and Communications (ICAIECC), 2014 International Conference on*, pages 1–4, Oct 2014.
- [TGG08] *Green Grid Data Center Power Efficiency Metrics: PUE and DCIE*, The Green Grid. The Green Grid, 2008. Accessed at: June 2014, [http://www.eni.com/green-data-center/it\\_IT/static/pdf/Green\\_Grid\\_DC.pdf](http://www.eni.com/green-data-center/it_IT/static/pdf/Green_Grid_DC.pdf).
- [Tra13] Harald Tranninger. Application specific Power Estimation for Virtualized Data Centers. Master’s thesis, Institute for Technical Informatics, Graz University of Technology, 2013.
- [Vad11] Abhay Vadher. *Papillon Architecture Specification Requirement*. Strategia Ltd., 2011.
- [Vad14] Abhay Vadher. *PAPILLON Power Model Generation - PROCESS / METHODOLOGY*. Document Number: SL-V1.0-009. Strategia Ltd., 2014.
- [VGP<sup>+</sup>13] S. Varrette, M. Guzek, V. Plugaru, X. Besson, and P. Bouvry. Hpc performance and energy-efficiency of xen, kvm and vmware hypervisors. In *Computer Architecture and High Performance Computing (SBAC-PAD), 2013 25th International Symposium on*, pages 89–96, Oct 2013.
- [WM07] Thomas Wiedmann and Jan Minx. *ECOLOGIAL ECONOMICS*. Research Trends. Nova Science Publishers, Inc., 2007. ISBN 978-1-60692-747-2.
- [ZK13] Rüdiger Zarnekow and Lutz Kolbe. *Green IT. Erkenntnisse und Best Practices aus Fallstudien*. Springer Gabler, Springer-Verlag Berlin Heidelberg 2013, 2013. ISBN 978-3-642-36151-7.